Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Специальность: «Программное обеспечение информационных технологий»

**КОНТРОЛЬНАЯ РАБОТА**

По курсу: «Современные платформы программирования»

На тему: «Ведение заказов»

Студент-заочник 2 курса

Группы № 581072

ФИО: Скоп Дмитрий Александрович

Адрес г. Минск,

ул. Уборевича д.38, кв. 37

Тел. +375293386519

Проверила: Бакунова О.М.

Минск, 2017

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc481768763)

[1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc481768764)

[1.1 Обзор существующих аналогов 7](#_Toc481768765)

[1.2 Постановка задачи 7](#_Toc481768766)

[1.3 Входные данные 8](#_Toc481768767)

[1.4 Выходные данные 8](#_Toc481768768)

[2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 9](#_Toc481768769)

[2.1 Система ERWin 9](#_Toc481768770)

[2.2 Диаграммы DFD 10](#_Toc481768771)

[3 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ 13](#_Toc481768772)

[3.1 Диаграмма вариантов использования 13](#_Toc481768773)

[3.2 Диаграмма классов 14](#_Toc481768774)

[3.3 База данных 16](#_Toc481768775)

[3.4 Выбор языка программирования и фреймворков 18](#_Toc481768776)

[3.5 Выбор среды разработки 19](#_Toc481768777)

[3.6 Выбор системы сборки 20](#_Toc481768778)

[4 ТЕСТИРОВАНИЕ 21](#_Toc481768779)

[4.1 Виды тестирования 22](#_Toc481768780)

[4.2 Уровни тестирования 23](#_Toc481768781)

[4.2.1 Компонентное (модульное) тестирование 23](#_Toc481768782)

[4.2.2 Интеграционное тестирование 24](#_Toc481768783)

[4.2.3 Системное тестирование 25](#_Toc481768784)

[4.2.4 Приемочное тестирование 25](#_Toc481768785)

[4.3 Тестирование программного средства 26](#_Toc481768786)

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 28](#_Toc481768787)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 33](#_Toc481768788)

[Приложение А: CREATE-скрипты таблиц базы данных 34](#_Toc481768789)

[Приложение Б: Фрагмент кода программного средства 43](#_Toc481768790)

ВВЕДЕНИЕ

Современным миром управляет торговля. Каждый день мы покупаем и продаем различные товары и услуги. Кто-то продает лучше, кто-то хуже. В условиях большой конкуренции важно использовать любое доступное преимущество.

Система управления заказами позволяет быстро и с минимальными затратами автоматизировать бизнес-процессы обработки и выполнения заказов. Высокая ценность услуг компании является ключом к увеличению спроса на товары и услуги, причем компания должна давать клиенту больше, чем ожидает клиент, и чем способны дать конкуренты, чтобы добиться успеха. Значительное увеличение ценности услуг компании обеспечивается за счет внедрения программных решений, позволяющих упростить работу с клиентом и уделить больше внимания мелочам.

С помощью такой системы менеджер может отслеживать текущий статус заказа, а так же всегда «иметь под рукой» информацию по стоимости заказа и детали доставки. Благодаря налаженному процессу управления заказами экономится время, а значит – и деньги.

Любое программное средство реализуется с помощью какого-либо языка программирования и соответствующих фреймворков. Язык программирования – это некоторый базовый синтаксис (возможно со стандартными библиотеками), с помощью которого можно создавать приложения. Фреймворк же предоставляет программисту различные библиотеки, значительно упрощающие создание программ. Не существует одного языка программирования, который бы превосходил все остальные. Превосходство какого-либо языка программирования может проявляться только в контексте какой-либо задачи.

Java представляет собой язык программирования и платформу вычислений, которая была впервые выпущена Sun Microsystems в 1995 г. Java отличается быстротой, высоким уровнем защиты и надежностью. От портативных компьютеров до центров данных, от игровых консолей до суперкомпьютеров, используемых для научных разработок, от сотовых телефонов до сети Интернет — сейчас Java можно найти повсюду.

Java Runtime Environment (JRE) состоит из Java Virtual Machine (JVM), базовых классов платформы Java и вспомогательных библиотек платформы Java. JRE является областью программного обеспечения Java, используемой во время выполнения, т.е. единственным компонентом, который требуется для запуска Java-приложений.

Одной из сильных сторон виртуальной машины Java всегда была ее способность с легкостью оперировать несколькими вычислительными потоками. JVM оптимизирована для больших многоядерных машин, и она без проблем может управлять сотнями потоков. Благодаря этой способности, на JVM появились и другие языки — создаются кросс-компиляторы и эмуляторы, работающие поверх JVM.

Язык Java не был первым языком для написания кроссплатформенных приложений, но он стал самым популярным. Java-разработчики могут разрабатывать приложение на своем компьютере, а затем развернуть его на целевой платформе, будь то телефон или сервер. Если для компилятора доступны нужные библиотеки, код будет работать.

JVM была построена и оптимизирована под типизированный код со статическим контекстом, генерируемый javac компилятором, но со временем разработчики языков поняли, что JVM может запускать код написанный не только на языке Java. Если компилятор создает корректный Java байт код, для JVM не имеет разницы, на каком языке он был написан. Таким образом появились Haskell, Scala, Clojure.

Одними из самых важных и распространенных фреймворков для Java являются Spring и Hibernate.

Spring Framework обеспечивает комплексную модель разработки и конфигурации для современных бизнес-приложений на Java - на любых платформах. Ключевой элемент Spring - поддержка инфраструктуры на уровне приложения: основное внимание уделяется "водопроводу" бизнес-приложений, поэтому разработчики могут сосредоточиться на бизнес-логике без лишних настроек в зависимости от среды исполнения.

Hibernate — это бесплатная Java библиотека с открытым исходным кодом, представляющая собой инструмент объектно-реляционного отображения (object-relational mapping — ORM). Его основной задачей является преобразование данных реляционнной БД в объектно-ориентированные модели и обратно. Помимо этого фреймворк также предоставляет средства для автоматического построения запросов, поиска и извлечения данных.

Целью Hibernate является освобождение разработчика от 95-ти процентов выполнения задач по обеспечению хранения объектов в реляционной базе данных. Hibernate может быть не лучшим решением для приложений, в которых бизнес логика построена на вызове хранимых процедур при работе с БД. Он будет более полезен для приложений, в которых обработка данных выполняется, скажем так, на среднем уровне, с использованием объектно-ориентированных моделей.

Hibernate может быть с легкостью внедрен на любой стадии проекта, его можно использовать как в процессе проектирования классов и таблиц, так и для работы с уже существующей базой данных.

Hibernate не требует никаких особых интерфейсов или базовых классов для создания персистентного класса и его связи с таблицами в БД. Абсолютно любой класс или структура может выступить в этой роли.

Hibernate поддерживает «ленивую» инициализацию и множество стратегий выборок. Он не требует наличия специальных таблиц или полей в базе данных и генерирует большую часть SQL кода во время инициализации системы, а не во время выполнения. Hibernate обычно обеспечивает превосходную производительность по сравнению с прямым JDBC-кодом.

Для сборки проектов в большинстве случаев не очень удобно пользоваться простым компилятором. Простые проекты можно собрать в командной строке, но если собирать большие проекты с командной строки, то команда для сборки будет очень длинной. Для упрощения сборки используют инструменты для сборки проектов.

Для платформы Java существуют два основных инструмента для сборки: Ant и Maven. Maven - это инструмент для сборки Java проекта: компиляции, создания jar, создания дистрибутива программы, генерации документации.

Вся структура проекта описывается в файле pom.xml (POM – Project Object Model), который должен находиться в корневой папке проекта. Как правило, большинство популярных библиотек находятся в центральном репозитории, поэтому их можно прописывать сразу в раздел dependencies pom-файла. Составив описание, достаточно выполнить одну команду – и проект будет собран, а библиотеки – автоматически загружены из репозитория.

Таким образом, Java является очень удобным и функциональным инструментом и может быть использована для создания системы управления заказами.

# 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Рассматриваемая предметная область – магазин, занимающийся продажей элитного алкоголя.

Пользователями системы будут выступать менеджеры, которые будут осуществлять ведение клиентской базы и управление заказами.

Процесс создания заказа осуществляется следующим образом:

1. Покупатель обращается в магазин с желанием приобрести определенный товар или товары;
2. Если менеджер считает нужным, он сохраняет всю личную информацию, предоставленную покупателем, в базу данных;
3. Далее менеджер собирает корзину необходимых покупателю товаров и оформляет заказ;
4. В дальнейшем менеджер может отслеживать заказ и управлять им, а так же легко найти информацию о клиенте в случае, если необходимо с ним связаться для уточнения каких-либо деталей заказа.

В системе такого типа должен присутствовать каталог товаров с возможностью манипуляций с товарами.

## 1.1 Обзор существующих аналогов

Существует большое количество программных комплексов, тем или иным способом предоставляющих функции управления заказами.

Например, скрипты интернет-магазинов: OpenCart, osCommerce, Zen Cart, eCart и многие другие. Они предоставляют интерфейс интернет-магазина с одной стороны и систему управления заказами с другой.

Так же существуют более комплексные решения, такие как ERP-системы. Зачастую такие системы избыточны для небольших предприятий, а высокая стоимость делает их использование нецелесообразным.

## 1.2 Постановка задачи

В данной контрольной работе требуется реализовать систему учета и ведения заказов.

Процесс проектирования архитектуры программного обеспечения состоит в проектировании структуры всех его компонент, функционально связанных с решаемой задачей, включая сопряжения между ними и требования к ним.

Во время разработки архитектуры программного обеспечения выполняется его модульно-иерархическое построение.

Определим следующие требования к разрабатываемой системе:

* должна быть реализована возможность добавления, удаления и редактирования товаров (каталог);
* должна быть реализована возможность сохранения информации о клиентах;
* должен быть реализован функционал «корзины»;
* должна быть реализована возможность создания и управления заказами;
* должна быть реализована возможность удаленной работы с данными в архитектуре клиент-сервер;
* хранение данных должно быть обеспечено средствами СУБД PostgreSQL;

При работе пользователя с разрабатываемой информационной системой не должно возникать проблем, система должна обладать понятным программным интерфейсом.

## 1.3 Входные данные

Входными данными для системы являются:

- информация о товаре (бренд, наименование, варианты товара, их цены);

- информация о клиенте (Ф.И.О., контактные данные);

- информация о заказе (сумма скидки, стоимость доставки и т.д.)

- реквизиты менеджеров для доступа к системе (логин, пароль).

## 1.4 Выходные данные

Выходными данными системы являются:

- отчеты об ошибках

- новая запись «товар»

- новая запись «клиент»

# 2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

IDEF0 (Integration Definition for Function Modeling) – нотация описания бизнес-процессов. Основана на методологии SADT.

SADT (Structured Analysis and Design Technique, технология структурного анализа и проектирования) - графические обозначения и подход к описанию систем. Разработка SADT началась в 1969 году и была опробована на практике в компаниях различных отраслей (аэрокосмическая отрасль, телефония и т.д.). Публично появилась на рынке в 1975 г и получила очень широкое распространение в мире.

IDEF0 является результатом программы компьютеризации промышленности, которая была предложена ВВС США. Автоматизация деятельности предприятий потребовала соответствующих методик и инструментов. Перед тем, как разрабатывать программное обеспечение, необходимо было четко и понятно описать бизнес-процессы.

Идея IDEF0 лежит в том, что бизнес-процесс отображается в виде прямоугольника, в которой входят и выходят стрелки.

Для IDEF0 имеет значение сторона процесса и связанная с ней стрелка:

* слева входящая стрелка – вход, информация, которая будет преобразована в ходе выполнения процесса;
* справа исходящая стрелка – выход – преобразованная информация;
* сверху входящая стрелка – управление бизнес-процесса – информация, определяющая, как должно происходить преобразования входных данных в выходные;
* снизу входящая стрелка – механизм бизнес-процесса – то, что преобразовывает вход в выход: сотрудники или техника.

## 2.1 Система ERWin

ERWin - это CASE-средство для проектирования и документирования баз данных, которое позволяет создавать, документировать и сопровождать базы данных, хранилища и витрины данных. Модели данных помогают визуализировать структуру данных, обеспечивая эффективный процесс организации, управления и администрирования таких аспектов деятельности предприятия, как уровень сложности данных, технологий баз данных и среды развертывания.

ERwin предназначен для всех компаний, разрабатывающих и использующих базы данных, для администраторов баз данных, системных аналитиков, проектировщиков баз данных, разработчиков, руководителей проектов, — и позволяет управлять данными в процессе корпоративных изменений, а также в условиях стремительно изменяющихся технологий.

ERwin позволяет наглядно отображать сложные структуры данных. Удобная в использовании графическая среда системы упрощает разработку базы данных и автоматизирует множество трудоёмких задач, уменьшая сроки создания высококачественных и высокопроизводительных транзакционных баз данных и хранилищ данных. Продукт улучшает коммуникацию организации, обеспечивая совместную работу администраторов и разработчиков баз данных, многократное использование модели, а также наглядное представление комплексных активов данных в удобном для понимания и обслуживания формате.

## 2**.2 Диаграммы DFD**

DFD – это методология графического структурного анализа, описывающая внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ. DFD – это один из основных инструментов структурного анализа и проектирования информационных систем, существовавших до широкого распространения UML.

Структурные нотации по-прежнему широко и эффективно используются как в бизнес-анализе, так и в анализе информационных систем.

Контекстная диаграмма (диаграмма верхнего уровня) - это диаграмма, показывающая разрабатываемую автоматизированную информационную систему в коммуникации с внешней средой.

На рисунке 2.1 изображена DFD-диаграмма верхнего уровня для системы ведения заказов.

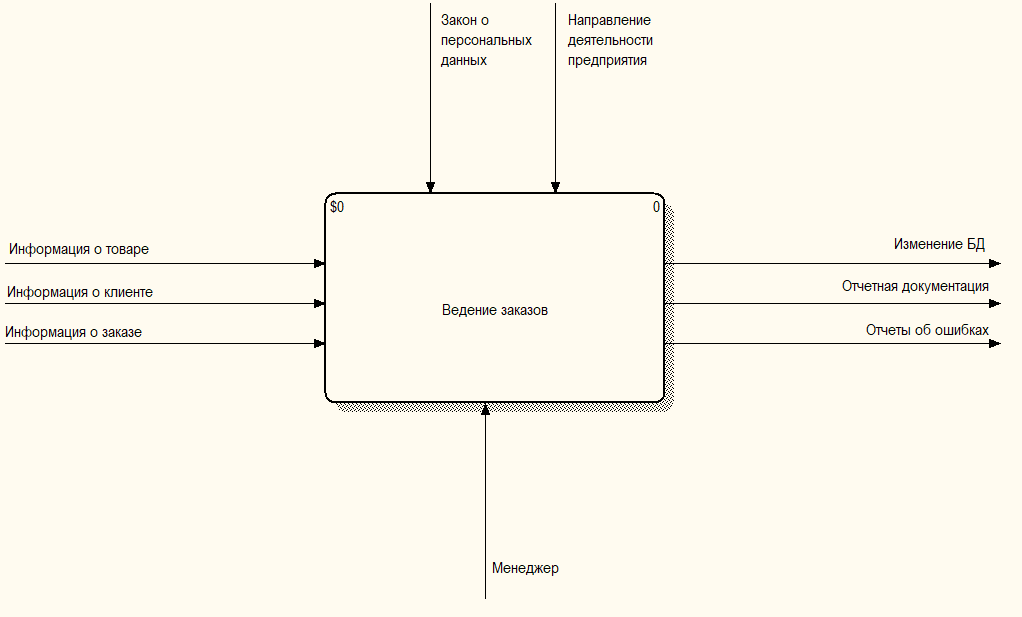


Рисунок 2.1– Диаграмма DFD, верхний уровень

Входными данными для системы являются: информация о товаре, личные данные клиента, информация о заказе.

Ограничениями являются: законодательство РБ, направление деятельности предприятия.

Пользователями системы является менеджер по продажам.

Результатом будет являться внесение в БД информации о товаре, клиенте или новом заказе.

На рисунке 2.2 изображена декомпозиция контекстной диаграммы.

Менеджер добавляет товар, после чего товар становится доступен для заказа. Затем менеджер вносит информацию о клиенте и собирает для него корзину товаров. Собрав корзину товаров, менеджер оформляет заказ.

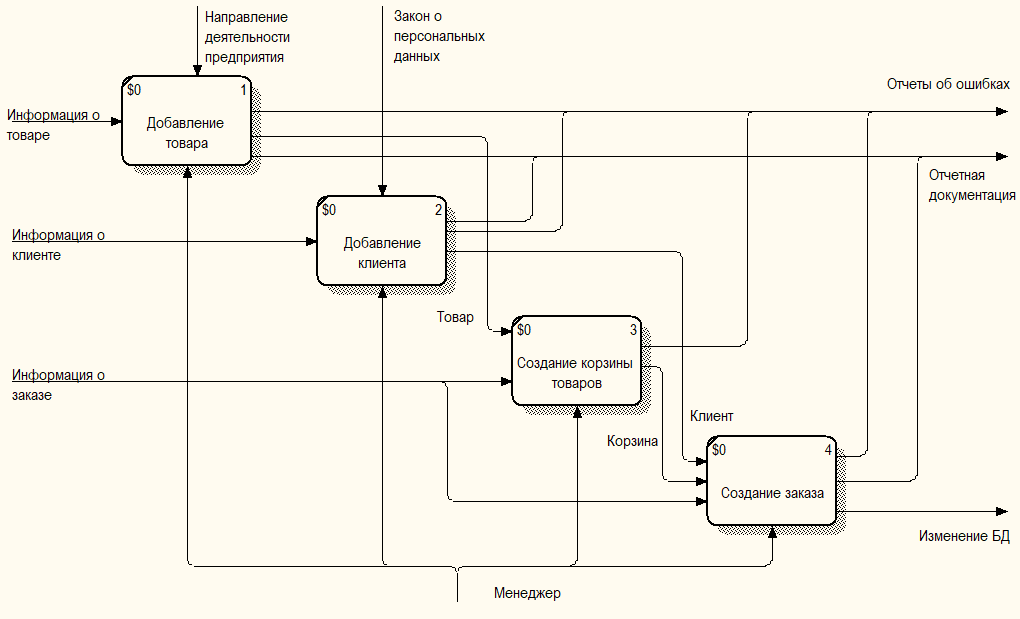


Рисунок 2.2 – Диаграмма DFD, уровень 0

# 3 ТЕХНИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

Основное назначение проектирования состоит в анализе структурных и функциональных отношений между элементами модели системы. Различные элементы логического представления, такие как классы, ассоциации, состояния, сообщения, не существуют материально или физически. Они лишь отражают наше понимание структуры физической системы или аспекты ее поведения.

В ходе проектирования архитектором или опытным программистом создается проектная документация, включающая текстовые описания, диаграммы, модели будущей программы.

Rational Rose представляет собой CASE средство проектирования и разработки информационных систем и программного обеспечения для управления предприятиями. Как и другие CASE средства (ARIS, BPwin, ERwin) его можно применять для анализа и моделирования бизнес процессов.

Принципиальное отличие Rational Rose от других средств заключается в объектно-ориентированном подходе. Графические модели, создаваемые с помощью этого средства, основаны на объектно-ориентированных принципах и языке UML (Unified Modeling Language). Инструменты моделирования Rational Rose позволяют разработчикам создавать целостную архитектуру процессов предприятия, сохраняя все взаимосвязи и управляющие воздействия между различными уровнями иерархии.

## 3.1 Диаграмма вариантов использования

UML (англ. Unified Modeling Language — унифицированный язык моделирования) — язык графического описания для объектного моделирования в области разработки программного обеспечения, моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это — открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода.

На рисунке 3.1 изображена диаграмма вариантов использования ПС.

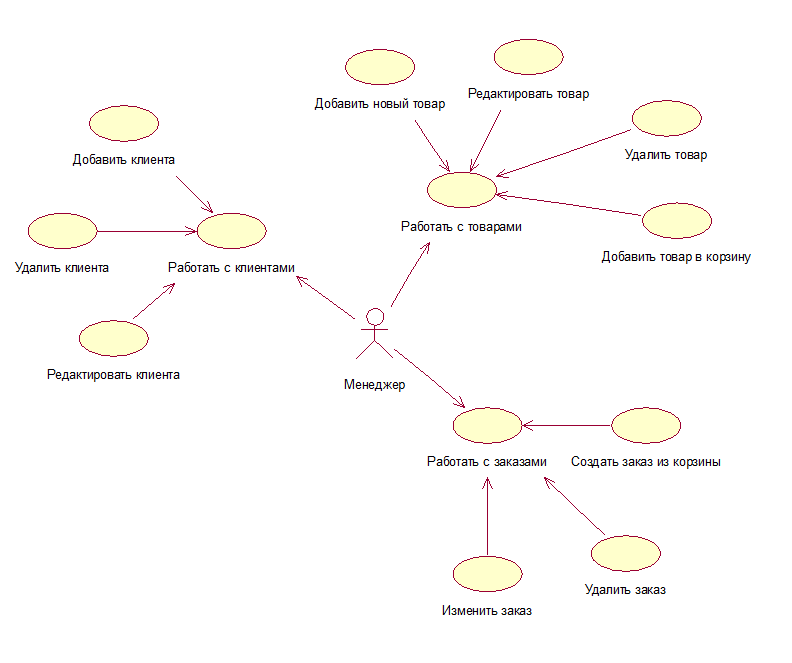


Рисунок 3.1 – Диаграмма вариантов использования

## 3.2 Диаграмма классов

Диаграмма классов— диаграмма, демонстрирующая классы системы, их атрибуты, методы и взаимосвязи между ними. Входит в UML.

Существует два вида:

- статический вид диаграммы рассматривает логические взаимосвязи классов между собой;

- аналитический вид диаграммы рассматривает общий вид и взаимосвязи классов, входящих в систему.

На рисунке 3.2 изображена диаграмма классов разрабатываемого ПС.

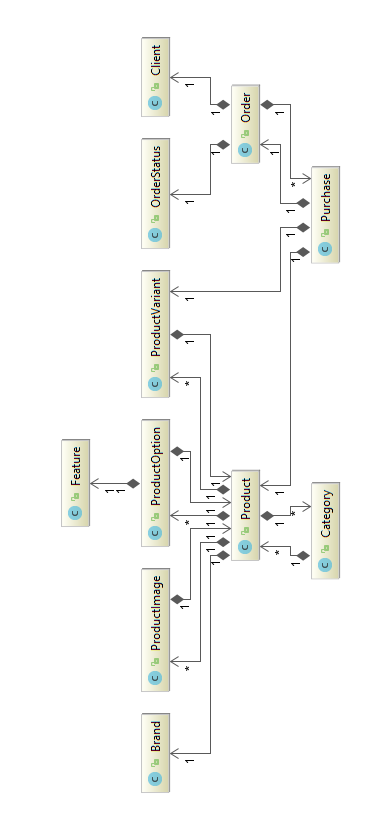


Рисунок 3.2 – Диаграмма классов

## 3.3 База данных

Для проектирования базы данных был использован инструмент DbSchema, который позволяет визуально проектировать базу данных.

Схема спроектированнной базы данных представлена на рисунке 2.3.

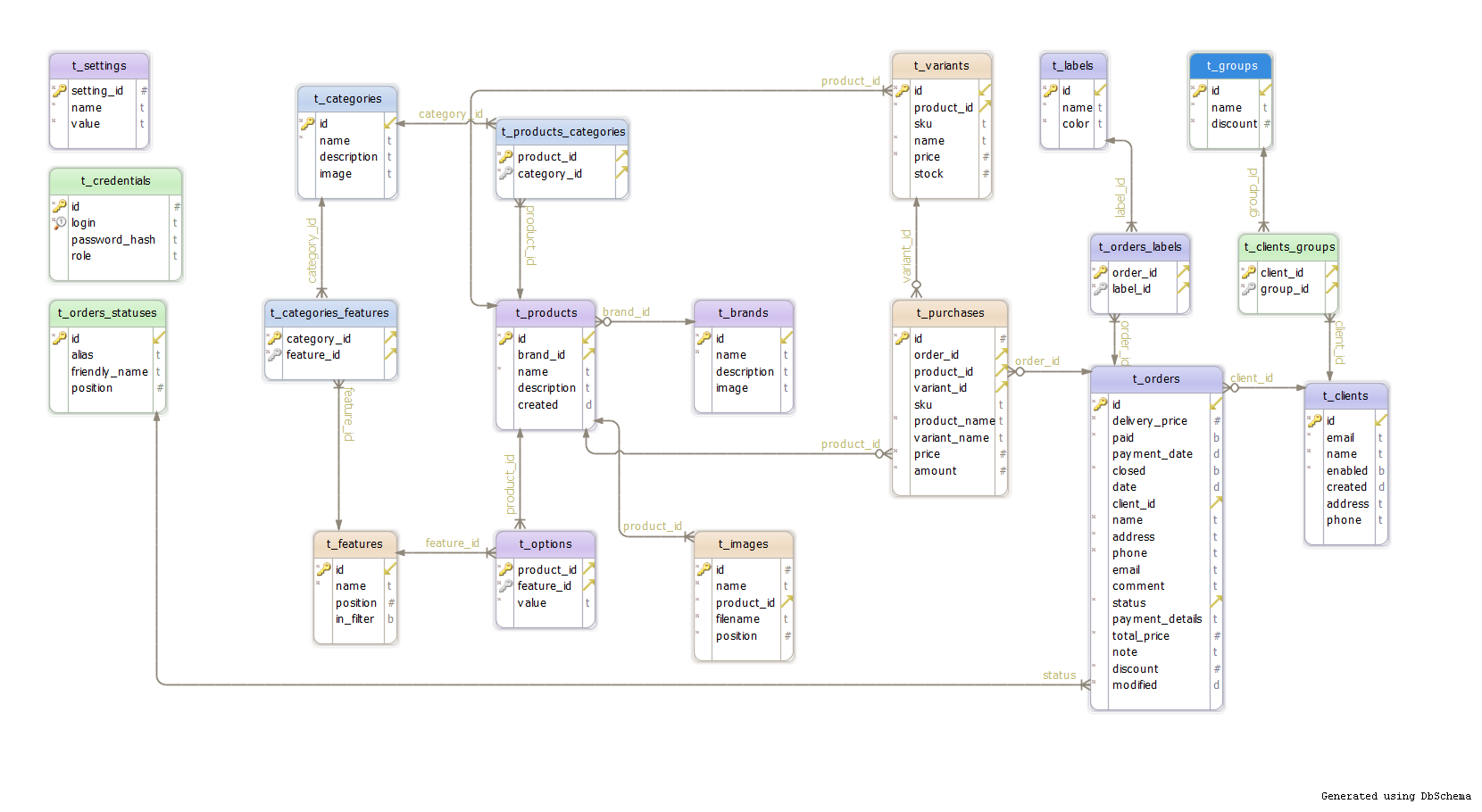


Рисунок 3.3 – Схема базы данных

В качестве СУБД была выбрана PostgreSQL-9.6.

PostgreSQL является объектно-реляционной системой управления базами данных (ОРСУБД) на основе POSTGRES, версия 4.2, разработанной в Университете Калифорнии в Беркли департаменте компьютерных наук.

PostgreSQL является open source потомком оригинального кода Berkeley. Он поддерживает большую часть стандарта SQL и предлагает множество современных функций:

- сложные запросы;

- внешние ключи;

- триггеры;

- представления;

- транзакционная целостность (transactional integrity);

- управление конкурентным доступом с помощью многоверсионности.

Кроме того, PostgreSQL может быть расширен пользователем во многих отношениях, например, путем добавления новых

- типов данных;

- функций;

- операторов;

- агрегатных функций;

- индекс методов;

- процедурных языков.

В составе сборки PostgreSQL идет множество полезных утилит:

initdb - инициализирует новое хранилище данных (кластер баз данных). Кластер представляет собой совокупность баз данных управляемых одним экземпляром севера. initdb должен быть запущен от имени будущего владельца сервера.

pg\_ctl - управляет процессом работы сервера PostgreSQL. Позволяет запускать, выполнять перезапуск, останавливать работу сервера, указать лог файл и другое.

psql - клиент для работы с базой данных. Позволяет выполнять SQL операции.

createdb - создает новую базу данных. По умолчанию, база данных создается от имени пользователя, который запускает команду. Чтобы задать другого пользователя — необходимо использовать опцию -O (если у пользователя есть необходимые привилегии для этого). По сути — это обертка SQL команды CREATE DATABASE.

dropdb - удаляет базу данных. Является оберткой SQL команды DROP DATABASE.

Createuser - добавляет нового пользователя базы дынных. Является оберткой SQL команды CREATE ROLE.

Dropuser - удаляет пользователя базы данных. Является оберткой SQL команды DROP ROLE.

Createlang - добавляет новый язык программирования в базу PostgreSQL. Является оберткой SQL команды CREATE LANGUAGE.

Droplang - удаляет язык программирования. Является оберткой SQL команды DROP LANGUAGE.

pg\_dump - создает бэкап (дамп) базы данных в файл.

pg\_restore - восстанавливает бэкап (дамп) базы данных из файла.

pg\_dumpall - создает бэкап (дамп) всего кластера в файл.

reindexdb - производит переиндексацию базы данных. Является оберткой SQL команды REINDEX.

clusterdb - производит перекластеризацию таблиц в базе данных. Является оберткой SQL команды CLUSTER.

vacuumdb - сборщик мусора и оптимизатор базы данных. Является оберткой SQL команды VACUUM.

## 3.4 Выбор языка программирования и фреймворков

В качестве языка программирования был выбран язык Java, а так же фреймворки Spring и Hibernate.

Java является кросс-платформенным языком, что является неоспоримым преимуществом при написании веб-приложений. Приложения, написанные на Java, будут работать на любом сервере с установленной средой Java Runtime Environment (JRE).

Spring это легковесный opensource J2EE Framework, разработка которого началась в феврале 2003 года. Основой послужил программный код, опубликованный в книге Expert One-on-One Design and Development (2002 год). Причем основные идеи возникли еще в 2000 году и являлись отражением опыта успешной разработки нескольких коммерческих проектов.

Сейчас Spring это достаточно популярный opensource проект, охватывающий многие аспекты как J2EE, так и Java разработок. В основе Spring лежит паттерн Inversion of control. Применительно к легковесным контейнерам, основная идея этого паттерна заключается в устранении зависимости компонентов или классов приложения от конкретных реализаций вспомогательных интерфейсов и делегировании полномочий по управлению созданием нужных реализаций IoC контейнеру.

Основные преимущества IoC контейнеров:

- управление зависимостями;

- упрощение повторного использования классов или компонентов;

- упрощение unit-тестирования;

- более "чистый" код (Классы больше не занимаются инициализацией вспомогательных объектов).

Hibernate — библиотека для языка программирования Java, предназначенная для решения задач объектно-реляционного отображения (ORM).

Целью Hibernate является освобождение разработчика от значительного объёма сравнительно низкоуровневого программирования при работе в объектно-ориентированных средствах в реляционной базе данных.

Mapping (сопоставление, проецирование) Java-классов с таблицами базы данных осуществляется с помощью конфигурационных XML-файлов и Java-аннотаций. При использовании файла XML Hibernate может генерировать скелет исходного кода для классов длительного хранения. В этом нет необходимости, если используется аннотация. Hibernate может использовать файл XML и аннотации для поддержки схемы базы данных.

В Hibernate обеспечиваются возможности по организации отношения между классами «один-ко-многим» и «многие-ко-многим». В дополнение к управлению связями между объектами Hibernate также может управлять рефлексивными отношениями, где объект имеет связь «один-ко-многим» с другими экземплярами своего собственного типа данных.

## 3.5 Выбор среды разработки

На момент написания данной работы одной из лучших сред разработки для Java является JetBrains IntelliJ IDEA.

Начиная с шестой версии продукта IntelliJ IDEA предоставляет интегрированный инструментарий для разработки графического пользовательского интерфейса. Среди прочих возможностей, среда хорошо совместима со многими популярными свободными инструментами разработчиков, такими как CVS, Subversion, Apache Ant, Maven и JUnit. В феврале 2007 года разработчики IntelliJ анонсировали раннюю версию плагина для поддержки программирования на языке Ruby.

Начиная с версии 9.0, среда доступна в двух редакциях: Community Edition и Ultimate Edition. Community Edition является полностью свободной версией, доступной под лицензией Apache 2.0, в ней реализована полная поддержка Java SE, Groovy, Scala, а также интеграция с наиболее популярными системами управления версиями. В редакции Ultimate Edition реализована поддержка Java EE, UML-диаграмм, подсчёт покрытия кода, а также поддержка других систем управления версиями, языков и фреймворков.

IntelliJ IDEA полностью совместима с выбранными для реализации программного средства фреймворками, что значительно ускоряет разработку.

## 3.6 Выбор системы сборки

Для упрощения сборки проекта был выбран инструмент Maven. Apache Maven — фреймворк для автоматизации сборки проектов на основе описания их структуры в файлах на языке POM (англ. Project Object Model).

Maven обеспечивает декларативную, а не императивную (в отличие от средства автоматизации сборки Apache Ant) сборку проекта. В файлах описания проекта содержится его спецификация, а не отдельные команды выполнения. Все задачи по обработке файлов, описанные в спецификации, Maven выполняет посредством их обработки последовательностью встроенных и внешних плагинов.

Maven используется для построения и управления проектами, написанными на Java, C#, Ruby, Scala, и других языках

# 4 ТЕСТИРОВАНИЕ

Тестирование программного продукта – это процесс исследования, испытания программного продукта, имеющий две различные цели:

- продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;

- выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

В более широком смысле, тестирование - это одна из техник контроля качества, включающая в себя активности по планированию работ (Test Management), проектированию тестов (Test Design), выполнению тестирования (Test Execution) и анализу полученных результатов (Test Analysis).

Стандарт ISO/IEC 25010:2011 (ГОСТ Р ИСО/МЭК 25010-2015) определяет качество программного обеспечения как степень удовлетворения системой заявленных и подразумеваемых потребностей различных заинтересованных сторон. В соответствии со стандартом модель качества продукта включает восемь характеристик:

- функциональная пригодность;

- уровень производительности;

- совместимость;

- удобство пользования;

- надёжность;

- защищённость;

- сопровождаемость;

- переносимость (мобильность).

Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны. Тестирование таких продуктов проводилось строго формализованно с записью всех тестовых процедур, тестовых данных, полученных результатов. Тестирование выделялось в отдельный процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

## 4.1 Виды тестирования

Существует несколько признаков, по которым принято производить классификацию видов тестирования. В зависимости от преследуемых целей, можно условно разделить виды тестирования на следующие группы:

- функциональные;

- нефункциональные;

- связанные с изменениями.

Функциональные тесты базируются на функциях и особенностях, а также взаимодействии с другими системами, и могут быть представлены на всех уровнях тестирования: компонентном или модульном (Component/Unit testing), интеграционном (Integration testing), системном (System testing) и приемочном (Acceptance testing). Функциональные виды тестирования рассматривают внешнее поведение системы. Далее перечислены одни из самых распространенных видов функциональных тестов:

- функциональное тестирование (Functional testing);

- тестирование безопасности (Security and Access Control Testing);

- тестирование взаимодействия (Interoperability Testing).

Нефункциональное тестирование описывает тесты, необходимые для определения характеристик программного обеспечения, которые могут быть измерены различными величинами. В целом, это тестирование того, "Как" система работает. Далее перечислены основные виды нефункциональных тестов:

- нагрузочное тестирование (Performance and Load Testing);

- стрессовое тестирование (Stress Testing);

- тестирование стабильности или надежности (Stability / Reliability Testing);

- объемное тестирование (Volume Testing);

- тестирование установки (Installation testing);

- тестирование удобства пользования (Usability Testing);

- тестирование на отказ и восстановление (Failover and Recovery Testing);

- конфигурационное тестирование (Configuration Testing).

Первые четыре из перечисленных можно объединить понятием «Тестирование производительности».

После проведения необходимых изменений, таких как исправление бага/дефекта, программное обеспечение должно быть перетестировано для подтверждения того факта, что проблема была действительно решена. Ниже перечислены виды тестирования, которые необходимо проводить после установки программного обеспечения, для подтверждения работоспособности приложения или правильности осуществленного исправления дефекта:

- дымовое тестирование (Smoke Testing);

- регрессионное тестирование (Regression Testing);

- тестирование сборки (Build Verification Test);

- санитарное тестирование или проверка согласованности/исправности (Sanity Testing).

## 4.2 Уровни тестирования

Тестирование на разных уровнях производится на протяжении всего жизненного цикла разработки и сопровождения программного обеспечения. Уровень тестирования определяет то, над чем производятся тесты: над отдельным модулем, группой модулей или системой, в целом. Проведение тестирования на всех уровнях системы - это залог успешной реализации и сдачи проекта.

Выделяют следующие уровни тестирования:

- компонентное или модульное тестирование (Component Testing or Unit Testing);

- интеграционное тестирование (Integration Testing);

- системное тестирование (System Testing);

- приемочное тестирование (Acceptance Testing).

### 4.2.1 Компонентное (модульное) тестирование

Компонентное (модульное) тестирование проверяет функциональность и ищет дефекты в частях приложения, которые доступны и могут быть протестированы по-отдельности (модули программ, объекты, классы, функции и т.д.). Обычно компонентное (модульное) тестирование проводится, вызывая код, который необходимо проверить, и при поддержке сред разработки, таких как фреймворки (frameworks - каркасы) для модульного тестирования или инструменты для отладки. Все найденные дефекты, как правило исправляются в коде без формального их описания в системе менеджмента багов (Bug Tracking System).

Один из наиболее эффективных подходов к компонентному (модульному) тестированию - это подготовка автоматизированных тестов до начала основного кодирования (разработки) программного обеспечения. Это называется разработка от тестирования (test-driven development) или подход тестирования вначале (test first approach). При этом подходе создаются и интегрируются небольшие куски кода, напротив которых запускаются тесты, написанные до начала кодирования. Разработка ведется до тех пор пока все тесты не будут успешно пройдены.

По сути, компонентный и модульный уровни тестирования представляют одно и тоже. Разница лишь в том, что в компонентном тестировании в качестве параметров функций используют реальные объекты и драйверы, а в модульном тестировании - конкретные значения.

### 4.2.2 Интеграционное тестирование

Интеграционное тестирование предназначено для проверки связи между компонентами, а также взаимодействия с различными частями системы (операционной системой, оборудованием либо связи между различными системами).

Уровни интеграционного тестирования:

Компонентный интеграционный уровень (Component Integration testing) - проверяется взаимодействие между компонентами системы после проведения компонентного тестирования.

Системный интеграционный уровень (System Integration Testing) – проверяется взаимодействие между разными системами после проведения системного тестирования.

Подходы к интеграционному тестированию:

Снизу вверх (Bottom Up Integration) – все низкоуровневые модули, процедуры или функции собираются воедино и затем тестируются. После чего собирается следующий уровень модулей для проведения интеграционного тестирования. Данный подход считается полезным, если все или практически все модули, разрабатываемого уровня, готовы. Также данный подход помогает определить по результатам тестирования уровень готовности приложения.

Сверху вниз (Top Down Integration) – вначале тестируются все высокоуровневые модули, и постепенно один за другим добавляются низкоуровневые. Все модули более низкого уровня симулируются заглушками с аналогичной функциональностью, затем по мере готовности они заменяются реальными активными компонентами. Таким образом мы проводим тестирование сверху вниз.

Большой взрыв ("Big Bang" Integration) – все или практически все разработанные модули собираются вместе в виде законченной системы или ее основной части, и затем проводится интеграционное тестирование. Такой подход очень хорош для сохранения времени. Однако, если тест кейсы и их результаты записаны не верно, то сам процесс интеграции сильно осложнится, что станет преградой для команды тестирования при достижении основной цели интеграционного тестирования.

### 4.2.3 Системное тестирование

Основной задачей системного тестирования является проверка как функциональных, так и не функциональных требований в системе в целом. При этом выявляются дефекты, такие как неверное использование ресурсов системы, непредусмотренные комбинации данных пользовательского уровня, несовместимость с окружением, непредусмотренные сценарии использования, отсутствующая или неверная функциональность, неудобство использования и т.д. Для минимизации рисков, связанных с особенностями поведения в системы в той или иной среде, во время тестирования рекомендуется использовать окружение максимально приближенное к тому, на которое будет установлен продукт после выдачи.

### 4.2.4 Приемочное тестирование

Приемочное тестирование – это формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью:

- определения удовлетворяет ли система приемочным критериям;

- вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет.

Приемочное тестирование выполняется на основании набора типичных тестовых случаев и сценариев, разработанных на основании требований к данному приложению.

Решение о проведении приемочного тестирования принимается, когда:

- продукт достиг необходимого уровня качества;

- заказчик ознакомлен с Планом Приемочных Работ (Product Acceptance Plan) или иным документом, где описан набор действий, связанных с проведением приемочного тестирования, дата проведения, ответственные и т.д.

Фаза приемочного тестирования длится до тех пор, пока заказчик не выносит решение об отправлении приложения на доработку или выдаче приложения.

## 4.3 Тестирование программного средства

Испытание клиентской части программы проводилось на аппаратных платформах, описание которых приведено в таблице 4.1.

Таблица 4.1 – Аппаратные платформы

|  |  |  |  |
| --- | --- | --- | --- |
| № | Роль | Аппаратная конфигурация | Программная конфигурация |
| 1 | Cервер базы данных, веб-сервер | Intel Core i5 4460, 128Gb SSD, 12Gb RAM | Microsoft Windows 10, PostgreSQL-9.6, Apache Tomcat 8.5 |
| 2 | Клиент | Intel Core i5 4460, 128Gb SSD, 12Gb RAM | Microsoft Windows 10, браузер Google Chrome |

Для проведения тестирования разработанного в рамках контрольной работы программного средства были написаны тест-кейсы, тестирующие корректность работы программы.

В таблице 4.2 приведен набор тест-кейсов, на основе которых производилось тестирование.

| Тестовый вариант | Входные данные | Ожидаемый результат |
| --- | --- | --- |
| 1 | 2 | 3 |
| Вход в систему | Правильные логин и пароль | Открытие страницы с заказами |
| Вход в систему с неправильными реквизитами | Несуществующая в БД комбинация логина и пароля | Возврат на страницу входа в систему |
| Добавление товара | Полные данные о товаре | Товар добавлен |
| Добавление товара с неполными данными | Неполные данные о товаре | Сообщение об ошибке |
| Удаление товара | ID товара | Товар удален |
| Открытие страницы товара | ID товара | Открыта страница с товаром |
| Редактирование товара | Новые данные о товаре | Товар отредактирован |
| Добавление клиента | Личные данные клиента | Клиент добавлен |
| Добавление клиента с неполными данными | Неполные данные о клиенте | Сообщение об ошибке |
| Удаление клиента | ID клиента | Клиент удален |
| Добавление товара в корзину | ID товара | Товар добавлен в корзину |
| Создание заказа из корзины | Личные данные клиента, информация о заказе (доставка, комментарий, размер скидки) | Заказ создан |
| Изменение статуса заказа | ID статуса, ID заказа | Статус изменен |
| Удаление заказа | ID заказа | Заказ удален |

В результате тестирования было устранено множество ошибок, таких как:

- отсутствие валидации вводимых пользователем данных. Для предотвращения ошибок из-за ввода пользователем некорректных данных была использована специальная библиотека для валидации полей ввода.

- проблема с входом в систему из-за некорректного хэша пароля;

- невозможность сохранить клиента во время создания заказа.

По результатам тестирования программное средство имеет надлежащее качество.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для работы с программным средством в системе пользователя должен быть установлен браузер Google Chrome версии не ниже 58 или аналогичный.

Для начала работы с приложением требуется перейти на страницу логина (корневая страница относительно контекста) и ввести в поля имя пользователя и пароль (manager : test). Внешний вид страницы представлен на рисунке 5.1

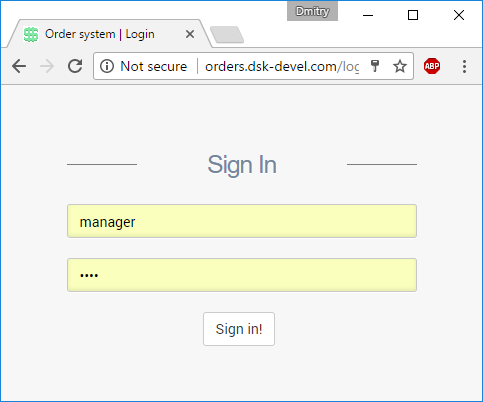


Рисунок 5.1 – Страница входа в систему

После входа в систему пользователь видит главный экран приложения.

Главный экран разделен на две части (рисунок 5.2). Левая часть – это меню, по которому осуществляется навигация. Правая часть – рабочая область. Меню может быть свернуто до минимизированного состояния. Для этого нужно кликнуть иконку «гамбургера».

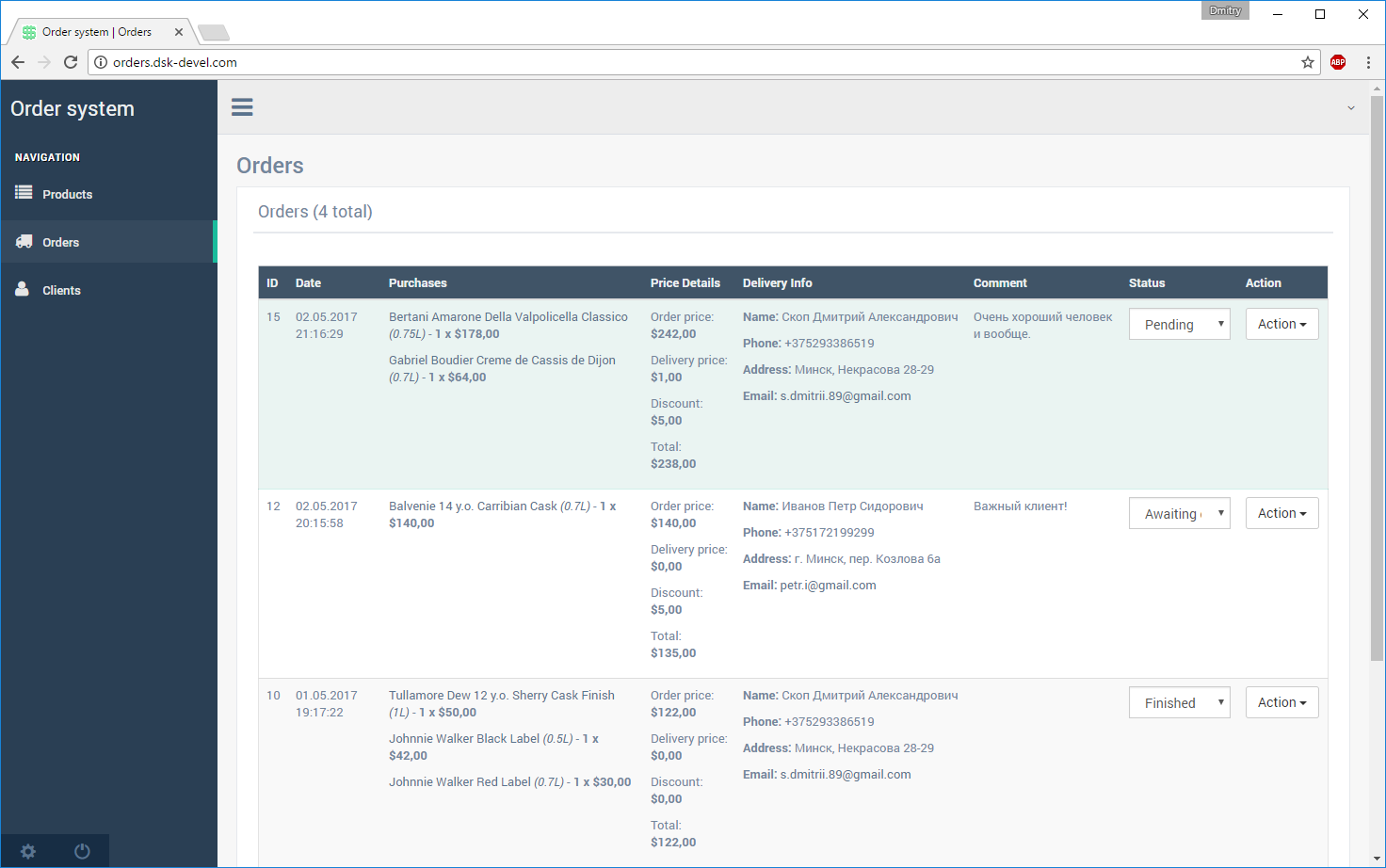


Рисунок 5.2 – Главный экран

Для работы с товарами необходимо перейти на вкладку «Products». На этой вкладке содержится форма добавления товара (рисунок 5.3), а так же таблица с уже существующими в системе товарами. Существующие товары выводятся постранично по 20 на страницу.

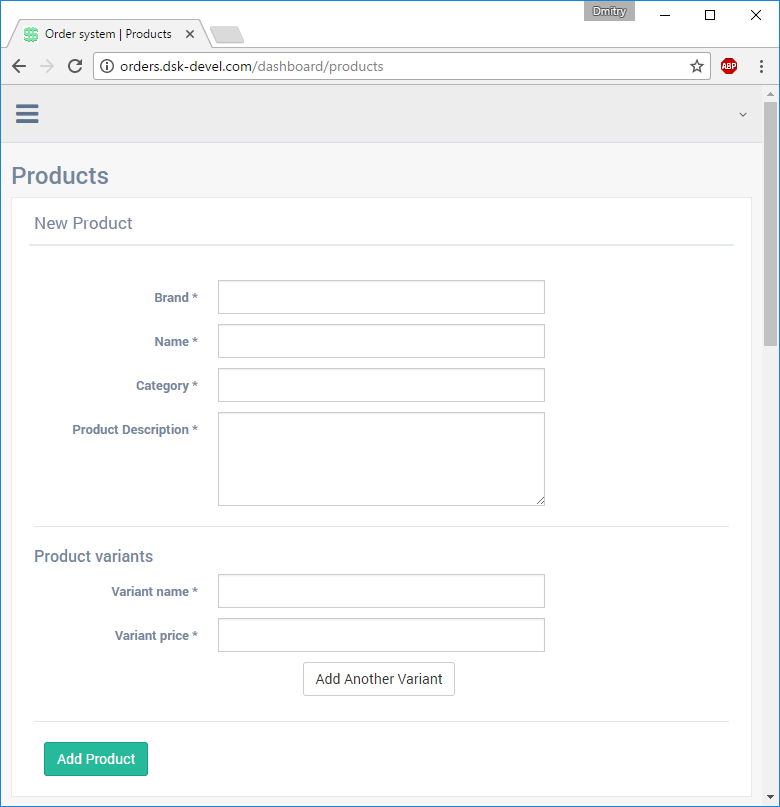


Рисунок 5.3 – Форма добавления товара

Во время добавления товара система будет выводить подсказки о уже существующих брендах и категориях.

Для просмотра и редактирования товара необходимо кликнуть на имя товара или выбрать Action -> View. Откроется страница товара, где можно произвести все необходимые манипуляции.

Чтобы добавить товар в корзину, необходимо кликнуть на ссылку с иконкой корзины «Add to cart». Товар будет добавлен в корзину, а в верхней части рабочей области появится иконка корзины (рисунок 5.4). Нажав эту иконку, можно увидеть содержимое корзины, а так же перейти к форме добавления заказа.

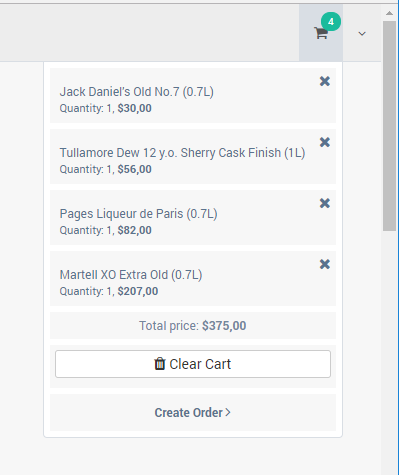


Рисунок 5.4 – Корзина

Для управления клиентами необходимо перейти на вкладку «Clients» (рисунок 5.5). На этой странице можно добавить нового клиента, а так же удалить ранее добавленных клиентов.

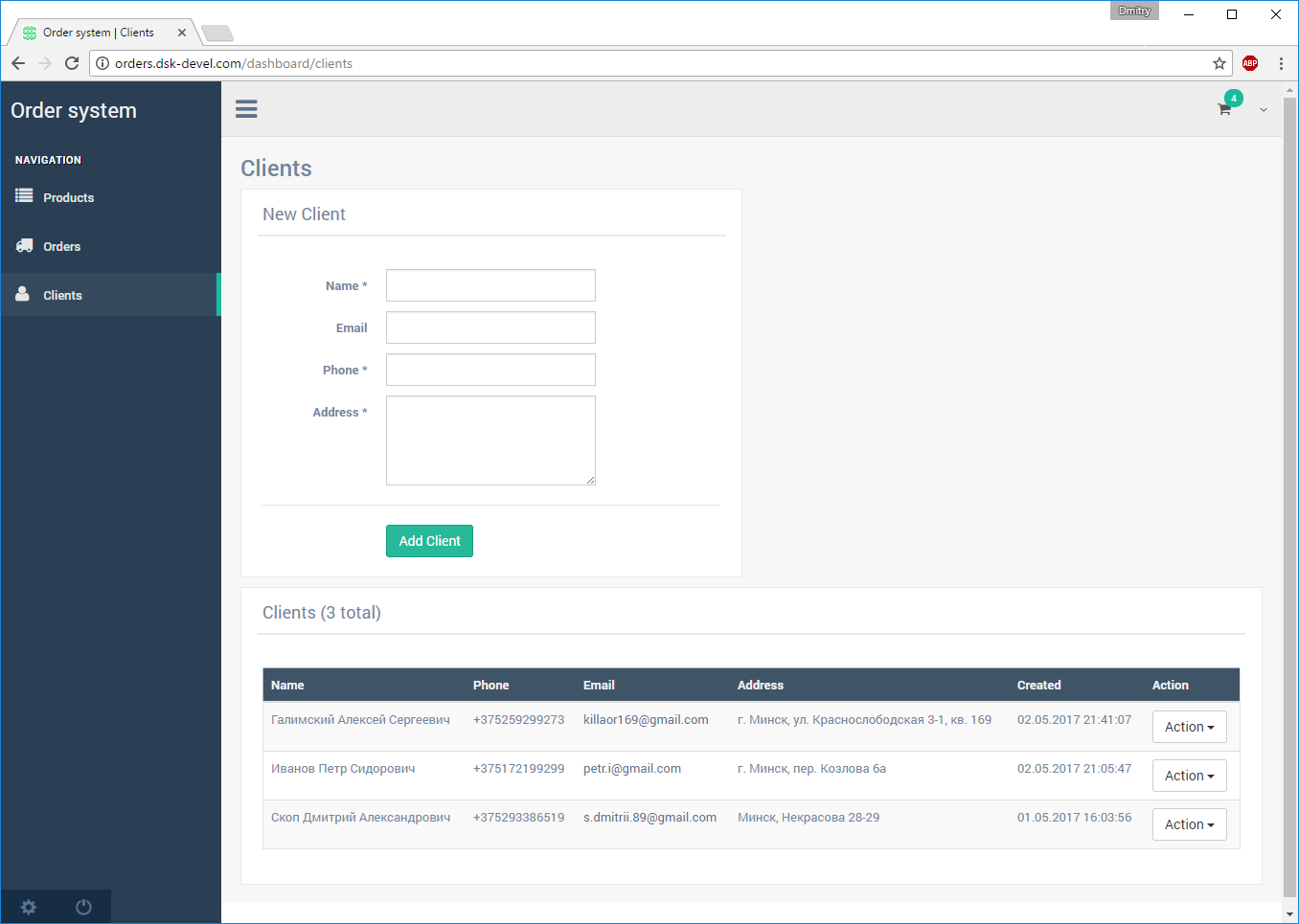


Рисунок 5.5 – Вкладка с клиентами

Для создания нового заказа, а так же управления уже добавленными в систему заказами, необходимо перейти на вкладку «Orders» (рисунок 5.6).

При вводе имени клиента система будет выдавать подсказку о уже существующих в базе клиентах. Если кликнуть на подсказку – форма с контактными данными для заказа будет автоматически заполнена.

Для изменения статуса уже добавленного заказа достаточно выбрать новый статус из выпадающего списка. Для удаления заказа нужно нажать Action -> Delete.

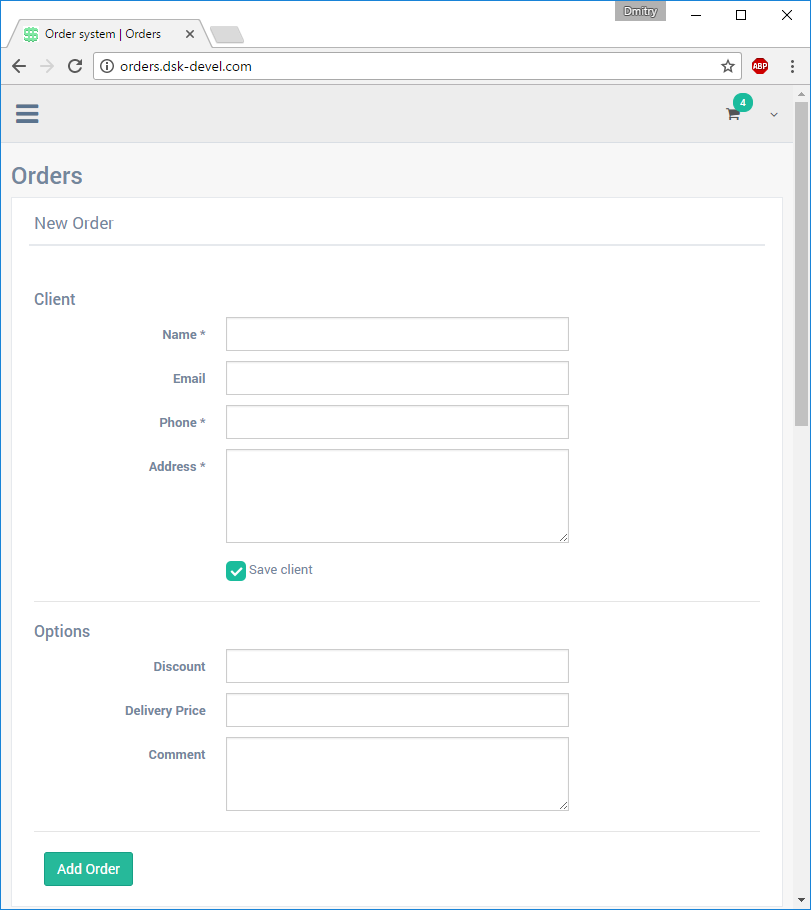


Рисунок 5.6 – Форма добавления заказа

Для выхода из системы необходимо кликнуть соответствующую иконку внизу левой панели (панели навигации) или выбрать Logout из выпадающего списка возле иконки корзины.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Леоненков. «Самоучитель UML».

2. Р. Мюллер. Базы данных и UML: Проектирование.– Лори, 2002г. 432 с.

3. Фельдман С.К. Система программирования Java без секретов: Как создать безопасное приложение с "нуля". – Новый издательский дом" , 2005 г. , 347 с.

4. Дейтел П.Дж., Дейтел Х.М. Как програмировать на Java. Книга 2. Файлы, сети, базы данных. – "Бином" · 2005 г., 672 с.

5. http://www.avacco.ru/page.asp?code=electronniy\_arhiv

6. <http://www.java.alfamoon.com/>

7. <https://ru.wikipedia.org/>

8. <http://www.protesting.ru>

9. <https://docs.spring.io/> - документация по фреймворку Spring

10.<http://docs.jboss.org/hibernate/orm/5.2/userguide/> - документация Hibernate ORM

# Приложение А: CREATE-скрипты таблиц базы данных

CREATE TABLE public.t\_brands

(

id integer NOT NULL DEFAULT nextval('t\_brands\_id\_seq'::regclass),

name character varying(255) NOT NULL DEFAULT ''::character varying,

description text,

image character varying(255),

CONSTRAINT t\_brands\_pkey PRIMARY KEY (id)

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_categories

(

id integer NOT NULL DEFAULT nextval('t\_categories\_id\_seq'::regclass),

name character varying(255) NOT NULL DEFAULT ''::character varying,

description text,

image character varying(255) DEFAULT ''::character varying,

CONSTRAINT t\_categories\_pkey PRIMARY KEY (id)

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_categories\_features

(

category\_id integer NOT NULL,

feature\_id integer NOT NULL,

CONSTRAINT t\_categories\_features\_pkey PRIMARY KEY (category\_id, feature\_id),

CONSTRAINT t\_categories\_features\_category\_id\_fkey FOREIGN KEY (category\_id)

REFERENCES public.t\_categories (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT t\_categories\_features\_feature\_id\_fkey FOREIGN KEY (feature\_id)

REFERENCES public.t\_features (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_clients

(

id integer NOT NULL DEFAULT nextval('t\_clients\_id\_seq'::regclass),

email character varying(255) NOT NULL,

name character varying(255) NOT NULL DEFAULT ''::character varying,

enabled boolean NOT NULL DEFAULT true,

created timestamp without time zone DEFAULT now(),

address character varying(255),

phone character varying(255),

CONSTRAINT t\_clients\_pkey PRIMARY KEY (id)

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_credentials

(

id integer NOT NULL DEFAULT nextval('t\_credentials\_id\_seq'::regclass),

login character varying NOT NULL,

password\_hash character varying(64),

role character varying(255),

CONSTRAINT t\_credentials\_pkey PRIMARY KEY (id)

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_features

(

id integer NOT NULL DEFAULT nextval('t\_features\_id\_seq'::regclass),

name character varying(255) NOT NULL,

"position" integer DEFAULT 0,

in\_filter boolean,

CONSTRAINT t\_features\_pkey PRIMARY KEY (id)

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_groups

(

id integer NOT NULL DEFAULT nextval('t\_groups\_id\_seq'::regclass),

name character varying(255) NOT NULL DEFAULT ''::character varying,

discount numeric(5,2) NOT NULL DEFAULT 0.00,

CONSTRAINT t\_groups\_pkey PRIMARY KEY (id)

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_clients\_groups

(

client\_id integer NOT NULL,

group\_id integer NOT NULL,

CONSTRAINT t\_clients\_groups\_pkey PRIMARY KEY (client\_id, group\_id),

CONSTRAINT t\_clients\_groups\_client\_id\_fkey FOREIGN KEY (client\_id)

REFERENCES public.t\_clients (id) MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE,

CONSTRAINT t\_clients\_groups\_group\_id\_fkey FOREIGN KEY (group\_id)

REFERENCES public.t\_groups (id) MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE CASCADE

)

WITH (

OIDS = FALSE

);

CREATE TABLE public.t\_images

(

id integer NOT NULL DEFAULT nextval('t\_images\_id\_seq'::regclass),

name character varying(255) NOT NULL,

product\_id integer NOT NULL DEFAULT 0,

filename character varying(255) NOT NULL DEFAULT ''::character varying,

"position" integer NOT NULL DEFAULT 0,

CONSTRAINT t\_images\_pkey PRIMARY KEY (id),

CONSTRAINT t\_images\_product\_id\_fkey FOREIGN KEY (product\_id)

REFERENCES public.t\_products (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_labels

(

id integer NOT NULL DEFAULT nextval('t\_labels\_id\_seq'::regclass),

name character varying(255) NOT NULL,

color character varying(7) NOT NULL,

CONSTRAINT t\_labels\_pkey PRIMARY KEY (id)

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_options

(

product\_id integer NOT NULL,

feature\_id integer NOT NULL,

value character varying(1024) NOT NULL,

CONSTRAINT t\_options\_pkey PRIMARY KEY (product\_id, feature\_id),

CONSTRAINT t\_options\_feature\_id\_fkey FOREIGN KEY (feature\_id)

REFERENCES public.t\_features (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT t\_options\_product\_id\_fkey FOREIGN KEY (product\_id)

REFERENCES public.t\_products (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_orders

(

id integer NOT NULL DEFAULT nextval('t\_orders\_id\_seq'::regclass),

delivery\_price numeric(10, 2) NOT NULL DEFAULT 0.00,

paid boolean NOT NULL DEFAULT false,

payment\_date timestamp with time zone,

closed boolean NOT NULL DEFAULT false,

date timestamp with time zone DEFAULT now(),

client\_id integer,

name character varying(255) COLLATE pg\_catalog."default" NOT NULL,

address character varying(255) COLLATE pg\_catalog."default" NOT NULL,

phone character varying(255) COLLATE pg\_catalog."default" NOT NULL,

email character varying(255) COLLATE pg\_catalog."default",

comment character varying(1024) COLLATE pg\_catalog."default",

status integer NOT NULL DEFAULT 0,

payment\_details text COLLATE pg\_catalog."default",

total\_price numeric(10, 2) NOT NULL DEFAULT 0.00,

note character varying(1024) COLLATE pg\_catalog."default",

discount numeric(5, 2) NOT NULL DEFAULT 0.00,

modified timestamp with time zone NOT NULL DEFAULT now(),

CONSTRAINT t\_orders\_pkey PRIMARY KEY (id),

CONSTRAINT t\_orders\_client\_id\_fkey FOREIGN KEY (client\_id)

REFERENCES public.t\_clients (id) MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE SET NULL,

CONSTRAINT t\_orders\_status\_fkey FOREIGN KEY (status)

REFERENCES public.t\_orders\_statuses (id) MATCH SIMPLE

ON UPDATE CASCADE

ON DELETE SET NULL

)

WITH (

OIDS = FALSE

);

CREATE TABLE public.t\_orders\_labels

(

order\_id integer NOT NULL,

label\_id integer NOT NULL,

CONSTRAINT t\_orders\_labels\_pkey PRIMARY KEY (order\_id, label\_id),

CONSTRAINT t\_orders\_labels\_label\_id\_fkey FOREIGN KEY (label\_id)

REFERENCES public.t\_labels (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT t\_orders\_labels\_order\_id\_fkey FOREIGN KEY (order\_id)

REFERENCES public.t\_orders (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_orders\_statuses

(

id integer NOT NULL DEFAULT nextval('t\_orders\_statuses\_id\_seq'::regclass),

alias character varying(255),

friendly\_name character varying(255),

"position" integer DEFAULT 0,

CONSTRAINT t\_orders\_statuses\_pkey PRIMARY KEY (id)

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_products

(

id integer NOT NULL DEFAULT nextval('t\_products\_id\_seq'::regclass),

brand\_id integer,

name character varying(500) NOT NULL,

description text,

created timestamp with time zone DEFAULT now(),

CONSTRAINT t\_products\_pkey PRIMARY KEY (id),

CONSTRAINT t\_products\_brand\_id\_fkey FOREIGN KEY (brand\_id)

REFERENCES public.t\_brands (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE SET NULL

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_products\_categories

(

product\_id integer NOT NULL,

category\_id integer NOT NULL,

CONSTRAINT t\_products\_categories\_pkey PRIMARY KEY (product\_id, category\_id),

CONSTRAINT t\_products\_categories\_category\_id\_fkey FOREIGN KEY (category\_id)

REFERENCES public.t\_categories (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT t\_products\_categories\_product\_id\_fkey FOREIGN KEY (product\_id)

REFERENCES public.t\_products (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_purchases

(

id integer NOT NULL DEFAULT nextval('t\_purchases\_id\_seq'::regclass),

order\_id integer DEFAULT 0,

product\_id integer DEFAULT 0,

variant\_id integer,

sku character varying(255),

product\_name character varying(255) NOT NULL DEFAULT ''::character varying,

variant\_name character varying(255),

price numeric(10,2) NOT NULL DEFAULT 0.00,

amount integer NOT NULL DEFAULT 0,

CONSTRAINT t\_purchases\_pkey PRIMARY KEY (id),

CONSTRAINT t\_purchases\_order\_id\_fkey FOREIGN KEY (order\_id)

REFERENCES public.t\_orders (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE,

CONSTRAINT t\_purchases\_product\_id\_fkey FOREIGN KEY (product\_id)

REFERENCES public.t\_products (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE SET NULL,

CONSTRAINT t\_purchases\_variant\_id\_fkey FOREIGN KEY (variant\_id)

REFERENCES public.t\_variants (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE SET NULL

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_settings

(

setting\_id integer NOT NULL DEFAULT nextval('t\_settings\_setting\_id\_seq'::regclass),

name character varying(255) NOT NULL DEFAULT ''::character varying,

value text NOT NULL,

CONSTRAINT t\_settings\_pkey PRIMARY KEY (setting\_id)

)

WITH (

OIDS=FALSE

);

CREATE TABLE public.t\_variants

(

id integer NOT NULL DEFAULT nextval('t\_variants\_id\_seq'::regclass),

product\_id integer NOT NULL,

sku character varying(255),

name character varying(255) NOT NULL,

price numeric(14,2) NOT NULL DEFAULT 0.00,

stock integer,

CONSTRAINT t\_variants\_pkey PRIMARY KEY (id),

CONSTRAINT t\_variants\_product\_id\_fkey FOREIGN KEY (product\_id)

REFERENCES public.t\_products (id) MATCH SIMPLE

ON UPDATE CASCADE ON DELETE CASCADE

)

WITH (

OIDS=FALSE

);

# Приложение Б: Фрагмент кода программного средства

**package** com.dsk\_devel.controllers;  
  
**import** com.dsk\_devel.database.model.\*;  
**import** com.dsk\_devel.database.repositories.\*;  
**import** com.dsk\_devel.dto.CartItemDto;  
**import** com.dsk\_devel.services.CartService;  
**import** com.dsk\_devel.util.ContentType;  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.data.domain.Page;  
**import** org.springframework.data.domain.PageRequest;  
**import** org.springframework.data.domain.Sort;  
**import** org.springframework.stereotype.Controller;  
**import** org.springframework.web.bind.annotation.RequestMapping;  
**import** org.springframework.web.bind.annotation.RequestParam;  
**import** org.springframework.web.bind.annotation.ResponseBody;  
**import** org.springframework.web.servlet.ModelAndView;  
  
**import** javax.servlet.http.HttpSession;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
@Controller  
**public class** OrdersController **extends** AjaxController {  
  
 @Autowired  
 **private** OrderRepository **orderRepository**;  
  
 @Autowired  
 **private** ClientRepository **clientRepository**;  
  
 @Autowired  
 **private** OrderStatusRepository **orderStatusRepository**;  
  
 @Autowired  
 **private** PurchaseRepository **purchaseRepository**;  
  
 @Autowired  
 **private** ProductVariantRepository **productVariantRepository**;  
  
 @RequestMapping(**"/components/new\_order"**)  
 **public** ModelAndView newOrder(HttpSession httpSession) {  
 ModelAndView mv = **new** ModelAndView(**"components/add-order.jsp"**);  
  
 CartService cartService = CartController.*getCartService*(httpSession);  
  
 List<CartItemDto> items = cartService.getWholeCart();  
  
 **float** priceTotal = 0.0f;  
 **int** itemsTotal = 0;  
  
 **for** (CartItemDto cartItemDto: items) {  
 priceTotal+=cartItemDto.getQuantity()\*cartItemDto.getVariant().getPrice();  
 itemsTotal+=cartItemDto.getQuantity();  
 }  
  
 mv.addObject(**"items"**, items);  
 mv.addObject(**"itemsTotal"**, itemsTotal);  
 mv.addObject(**"priceTotal"**, priceTotal);  
  
 **return** mv;  
 }  
  
 @RequestMapping(**"/action/order/delete"**)  
 **public** String deleteOrder(@RequestParam(name = **"id"**) **int** orderId,  
 @RequestParam(name = **"returnTo"**, defaultValue = **"/"**) String returnTo) {  
  
 **orderRepository**.delete(orderId);  
  
 **return "redirect:"** + returnTo;  
 }  
  
 @RequestMapping(**"/components/orders\_list"**)  
 **public** ModelAndView ordersList(@RequestParam(name = **"page"**, defaultValue = **"0"**) **int** page) {  
 ModelAndView mv = **new** ModelAndView(**"components/orders-list.jsp"**);  
  
 Page<Order> ordersPage = **orderRepository**.findAll(**new** PageRequest(page, 20,  
 **new** Sort(**new** Sort.Order(**"status.position"**),  
 **new** Sort.Order(**"orderDate"**))));  
  
 List<OrderStatus> statuses = **new** ArrayList<>();  
  
 **orderStatusRepository**.findAll().forEach(statuses::add);  
  
  
 mv.addObject(**"orders"**, ordersPage.getContent());  
 mv.addObject(**"ordersTotal"**, ordersPage.getTotalElements());  
 mv.addObject(**"totalPages"**, ordersPage.getTotalPages());  
 mv.addObject(**"statuses"**, statuses);  
  
 **return** mv;  
 }  
  
 @RequestMapping(value = **"/ajax/order/set\_status"**, produces = ContentType.***JSON\_CONTENT\_TYPE***)  
 @ResponseBody  
 **public** String setOrderStatus(@RequestParam(name = **"statusId"**) **int** statusId,  
 @RequestParam(name = **"orderId"**) **int** orderId) {  
  
 Order order = **orderRepository**.findOne(orderId);  
 OrderStatus status = **orderStatusRepository**.findOne(statusId);  
  
 **if** (order != **null** && status != **null**) {  
 order.setStatus(status);  
 **orderRepository**.save(order);  
 **return** success();  
 } **else** {  
 **return** error(**"BAD\_ORDER\_OR\_STATUS\_ID"**);  
 }  
  
  
 }  
  
 @RequestMapping(**"/ajax/order/add"**)  
 @ResponseBody  
 **public** String addOrder(@RequestParam(name = **"clientJson"**) String clientJson,  
 @RequestParam(name = **"saveClient"**, defaultValue = **"true"**) **boolean** saveClient,  
 @RequestParam(name = **"discount"**, defaultValue = **"0"**) **float** discount,  
 @RequestParam(name = **"deliveryPrice"**, defaultValue = **"0"**) **float** deliveryPrice,  
 @RequestParam(name = **"comment"**, required = **false**) String comment,  
 HttpSession httpSession) {  
 CartService cartService = CartController.*getCartService*(httpSession);  
  
 Client fromJson = **GSON**.fromJson(clientJson, Client.**class**);  
 Client client = **null**;  
  
 *//Create order* Order order = **new** Order();  
  
 **if** (fromJson.getId() > 0) {  
 *//find client in db* client = **clientRepository**.findOne(fromJson.getId());  
 } **else** {  
 client = fromJson;  
 **if** (saveClient) {  
 **clientRepository**.save(client);  
 order.setClient(client);  
 }  
 }  
  
 order.setDeliveryAddress(fromJson.getAddress());  
 order.setDeliveryName(fromJson.getName());  
 order.setDeliveryPhone(fromJson.getPhone());  
 order.setDeliveryPrice(deliveryPrice);  
 order.setEmail(fromJson.getEmail());  
 order.setComment(comment);  
  
 OrderStatus orderStatus = **orderStatusRepository**.getByAlias(**"PENDING"**);  
 order.setStatus(orderStatus);  
  
 **orderRepository**.save(order);  
  
 **float** priceTotal = 0.0f;  
  
 **for** (CartItemDto cartItemDto: cartService.getWholeCart()) {  
 ProductVariant productVariant = **productVariantRepository**.findOne(cartItemDto.getVariant().getId());  
  
 **if** (productVariant != **null**) {  
 Purchase purchase = **new** Purchase();  
 purchase.setOrder(order);  
 purchase.setProduct(productVariant.getProduct());  
 purchase.setVariant(productVariant);  
  
 purchase.setAmount(cartItemDto.getQuantity());  
 purchase.setPrice(productVariant.getPrice());  
 purchase.setProductName(productVariant.getProduct().getName());  
 purchase.setVariantName(productVariant.getName());  
  
 **purchaseRepository**.save(purchase);  
  
 priceTotal+=productVariant.getPrice();  
 } **else** {  
 cartService.removeFromCart(cartItemDto.getVariant());  
 }  
 }  
 order = **orderRepository**.findOne(order.getId());  
  
 **if** (priceTotal > 0) {  
 order.setTotalPrice(priceTotal);  
 order.setDiscount(discount);  
 **orderRepository**.save(order);  
 } **else** {  
 **orderRepository**.delete(order);  
 }  
  
 cartService.clearCart();  
  
 **return** success();  
 }  
  
}