**Содержание**

[Введение 2](#_Toc54186612)

[1. Анализ предметной области и ее формализация для проектирования базы данных 3](#_Toc54186613)

[1.1. Описание предметной области 3](#_Toc54186614)

[1.2. Анализ информационных потребностей пользователей и предварительное описание запросов 5](#_Toc54186615)

[1.3. Определение требований и ограничений к базе данных с точки зрения предметной области 6](#_Toc54186616)

[1.4. Постановка решаемой задачи 6](#_Toc54186617)

[2. Проектирование базы данных для основного вида деятельности рассматриваемой предметной области 8](#_Toc54186618)

[2.1. Разработка инфологической модели предметной области базы данных 8](#_Toc54186619)

[2.2. Выбор и обоснование используемых типов, данных и ограничений (доменов) 9](#_Toc54186620)

[2.3. Проектирование запросов к базе данных 12](#_Toc54186621)

[2.4. Программная реализация и документирование базы данных 16](#_Toc54186622)

[2.5. Проектирование разрабатываемого приложения 18](#_Toc54186623)

[3.2. Администрирование базы данных 26](#_Toc54186624)

[3.3. Реализация клиентских запросов 28](#_Toc54186625)

[3.4. Обоснование и реализация механизма обеспечения безопасности и сохранности данных 34](#_Toc54186626)

[Заключение 38](#_Toc54186627)

[Список использованных источников 39](#_Toc54186628)

[Приложение А – Код страницы работы с товарами 40](#_Toc54186629)

[Приложение Б – Код обработки запросов сущности пользователь 44](#_Toc54186630)

# Введение

Важнейшая задача компьютерных систем – хранение и обработка данных. Для её решения были предприняты усилия, которые привели к появлению в конце 60-х – начале 70-х годов специализированного программного обеспечения – систем управления базами данных (database management systems) [1]. СУБД позволяют структурировать, систематизировать и организовать данные для их компьютерного хранения и обработки. Невозможно представить себе деятельность современного предприятия или учреждения без использования профессиональных СУБД. Несомненно, они составляют фундамент информационной деятельности во всех сферах – начиная с производства и заканчивая финансами и телекоммуникациями.

Курсовой проект посвящен изучению теории и практики разработки и проектирования распределённых баз данных. В данном курсовом проекте объектом исследования является среда WEB, как платформа приложений баз данных в информационных системах.

Предметом исследования является система организации и ведения распределённых баз данных в Интернет. Целью курсового проекта является разработка базы данных для поддержки работы интернет-магазина.

Курсовой проект предполагает выполнение следующих задач:

* спроектировать базу данных интернет-магазин, описать предметную область, создать инфологическую модель и запросы к БД;
* разработать с использованием Веб–интерфейса, алгоритм работы базы данных для поддержки работы интернет-магазин.

В качестве практической части в рамках курсового проекта создаётся база данных для поддержки работы интернет-магазин с использованием технологий языка программирования Java и СУБД MySQL. Выбранная тема считается актуальной на сегодняшний день, так как сегодня миллионы людей ежедневно, не выходя из дому, совершают покупки в интернете, благодаря чему количество сервисов, предоставляющих такую возможность растет.

# 1. Анализ предметной области и ее формализация для проектирования базы данных

## Описание предметной области

Интернет-магазин (e-shop) – это виртуальная торговая точка, расположенная в Интернете, доступная для посещения любому пользователю сети и дающая ему возможность дистанционно заказать тот или иной товар через Интернет [2].

Существует 3 способа создания Интернет-магазина:

Аренда Интернет-магазина – предоставление программной основы сайта-магазина в совокупности с сервисным обслуживанием.

Такой способ выгоден, если предприятие до конца не уверено в необходимости приобретения или сомневается в быстроте окупаемости такого проекта.

Несмотря на то, что аренда выглядит на первый взгляд более привлекательно, чем самостоятельное его создание, обычно арендованный магазин не содержит отличительных особенностей предприятия.

Покупка готового Интернет-магазина – приобретение размещённого в Интернете уникального решения Интернет-каталога.

Основное преимущество – подготовленность дизайна. Приобретая готовый сайт-магазин владелец экономит время и тратит деньги с полным осознанием того, что он покупает.

Разработка Интернет-магазина – настройка и кастомизация базового решения под специфику продажи товаров конкретного предприятия.

В этом случае формируется концепция и принципы работы магазина, программа его продвижения, требования к персоналу, а также разрабатывается эксклюзивный дизайн.

В зависимости от функциональных возможностей электронного магазина, работающий в режиме онлайн, выделяют три категории:

* интернет-витрина;
* торговый автомат;
* автоматический магазин.

В случае Интернет-витрины электронного магазина представляет собой логическое расширение обычного web-сайта, когда на него просто выкладывается вся необходимая информация о товарах, которая по мере надобности обновляется. По существу, Интернет-витирина реализует только рекламную функцию. Для осуществления покупки после посещения такого электронного магазина пользователь проходит обычный цикл: звонок или визит в компанию, оплата и доставка.

Торговый автомат, в отличие от Интернет-витрины, осуществляет реальную торговлю. Система ведения торговых операций некоторым образом интегрирована с бизнес-процессами организации. Информация о сделанных заказах может обрабатываться по разному: как автоматически, так и менеджером вручную. Электронный магазин этого типа является наиболее предпочтительным для пилотных и тестовых проектов с небольшим числом покупателей.

Автоматический магазин в общем случае позволяет в автоматическом режиме принимать заказ, выписывать счет, принимать платеж, а также формировать заявку на доставку товара покупателю. Степень интеграции торговых операций с бизнес-процессами высокая. Менеджер осуществляет лишь общий контроль за работой системы. Электронный магазин этого типа довольно сложны, требуют немалых затрат на разработку, но вместе с тем, могут существенно увеличить объемы продаж.

Полнофункциональный электронный магазин в общем случае содержит электронную витрину для просмотра и выбора товара, подсистему расчетов с покупателем и службу доставки товара. Ключевая роль в организации электронного магазина принадлежит подсистеме расчетов, которая взаимодействует с одной или несколькими платежными системами.

Все электронные платежные системы условно делятся на два вида: кредитные и дебетовые. Электронные платежные системы в свою очередь делятся на:

* анонимные (когда, в отличие от обычных наличных денег, невозможно проследить за переходом денег из рук в руки);
* неанонимные (когда либо только банк, либо еще и продавец может получить персональные и банковские данные клиента, включая детали покупки).

## 1.2. Анализ информационных потребностей пользователей и предварительное описание запросов

Так как целью работы является не создание интернет-магазина, а поддержка работы его базы данных, возможности создания пользователем корзины покупки с последующими операциями над ней был опущен.

Поскольку работать с базой будут работники магазина, правильнее будет разделить уровень доступа на 2 – менеджера и администратора, позволив первым работать с данными, касающимися продуктов, а вторым – расширить возможности первых и управлять данными персонала и пользователей.

Так или иначе, магазин работает с товарами, так что нужна возможность работать напрямую с ними. Кроме отдельного меню для создания / чтения / редактирования / удаления данных хорошей вещью будет создания отдельного запроса, по которому пользователь сможет быстро просматривать имеющийся товар по категории или бренду.

Кроме работы с товарами, необходимо реализовать ведение базы данных, содержащей информацию о пользователях и сотрудниках магазина. Для безопасности необходимо, чтобы работать с этими данными может лишь администратор.

Необходимо предусмотреть операции создания / чтения / редактирования / удаления для всех таблиц базы данных.

Поскольку интернет-магазины могут быть большими по объему персонала, лучшим вариантом будет предоставление возможности многопользовательского доступак базе данных

## 1.3. Определение требований и ограничений к базе данных с точки зрения предметной области

Ограничения к разрабатываемой базе данных следующие:

Таблица пользователей / сотрудников, включающая в себя поле электронного адреса имеет маску, которая не позволит ввести значение без символа @.

Таблиц продуктов, имеющая такие поля как цена и количество, имеет ограничение по вводимым значением, которое не позволит ввести вместо цифр буквы или любые другие лишние символы

Все поля базы данных должны быть заполнены, пустых полей не допускается.

## 1.4. Постановка решаемой задачи

Для достижения цели данного курсового проекта, следует выделить следующие задачи, которые необходимо решить:

* исследовать предметную область,
* проанализировать и обосновать выбор технологий для разработки интернет-сайта поддержки базы данных интернет-мазагина,
* разработать интуитивно-понятный и адаптивный дизайн, удобный для пользователей, учитывающий основные особенности навигации,

Кроме того, разрабатываемое приложение должно следующим требованиям

* проектируемая база данных должна отвечать требованиям надёжности, минимальной избыточности, целостности данных
* содержать не менее 5 сущностей и должна быть приведена к третьей нормальной форме
* база данных должна поддерживать основные современные средства для работы и администрирования.
* для проверки работоспособности разработанной базы данных необходимо разработать клиент-серверное приложение веб- или мобильное приложение.
* При реализации приложения необходимо использовать версии технологий, являющиеся актуальными и стабильными на 30.06.2020.
* реализовать приложение необходимо при помощи языка программирования Java и Базы данный SQL.

Разрабатываемая система должна поддерживать:

* работу с каталогом товаров (добавление, удаление, изменение и просмотр информации о товарах),
* работу с данными клиентов (добавление и удаление записей в БД).

# 2. Проектирование базы данных для основного вида деятельности рассматриваемой предметной области

## 2.1. Разработка инфологической модели предметной области базы данных

Основной сущностью, с которой работает интернет-магазин является товар (продукт). У этого товара так или иначе должно быть имя и цена, по которой он продается. Обычно товар находится не в единственном числе, так что должно быть описание количества выбранного товара. Сущность хоть и вкратце, но должна иметь описание товара. Еще товары делятся на категории и бренды.

Поскольку категорий и брендов много, а также к одной категории или к одному бренду могут относиться несколько товаров, то их также можно считать отдельными сущностями, состоящими из названия себя.

Для работы с людьми, такими как пользователь или сотрудник, будет использоваться соответствующая сущность. Она будет содержать ФИО человека, адрес его проживания, номер телефона, а также логин и пароль. Кроме того, для разграничения сущностей пользователя (людей) по категориями, будет использоваться отдельная сущность роль.

Сущность роль представляет из себя вариант роли, благодаря которому будет осуществляться разделение полномочий.

По полученной выше логической модели построим физическую модель для реляционной СУБД

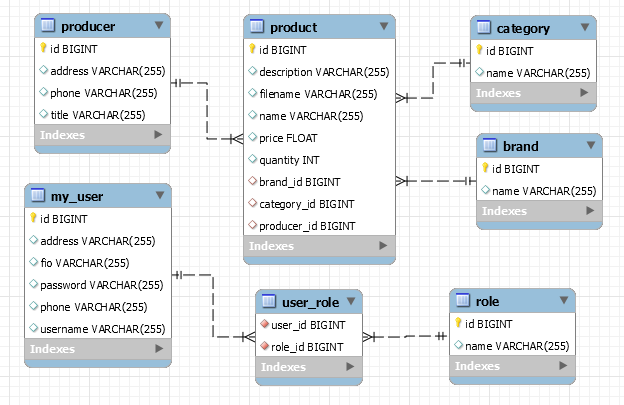


Рисунок 2.1 – нормализованная физическая структура базы данных

## 2.2. Выбор и обоснование используемых типов, данных и ограничений (доменов)

Созданная база данных состоит из семи таблиц. Описание таблиц базы данных приведено в таблице 2.1.

Таблица 2.1 – Состав таблиц базы данных

|  |  |
| --- | --- |
| Имя таблицы | Описание |
| producer | Справочник поставщиков |
| category | Справочник категорий |
| brand | Справочник брендов |
| product | Справочник товаров |
| role | Справочник ролей |
| my\_user | Справочник пользователей |
| user-role | Вспомогательная таблица для обеспечение связи многие ко многим |

В таблицах 2.2 – 2.8 приведено описание состава таблиц спроектированной базы данных.

Таблица 2.2 – producer

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Пустые значения (Да/Нет) |
| PK | Id | Число | Нет | Нет |
| Нет | Address | Строка | Нет | Нет |
| Нет | Phone | Строка | Нет | Нет |
| Нет | title | Строка | Нет | Нет |

Таблица 2.3 – my\_user

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Пустые значения (Да/Нет) |
| PK | Id | Число | Нет | Нет |
| Нет | Address | Строка | Нет | Нет |
| Нет | Fio | Строка | Нет | Нет |
| Нет | Password | Строка | Нет | Нет |
| Нет | Phone | Строка | Нет | Нет |
| Нет | username | Строка | Нет | Нет |

Таблица 2.4 – product

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Пустые значения (Да/Нет) |
| PK | Id | Число | Нет | Нет |
| Нет | Description | Строка | Нет | Нет |
| Нет | Filename | Строка | Нет | Нет |
| Нет | Name | Строка | Нет | Нет |
| Нет | Price | Число | Нет | Нет |
| Нет | Quantity | Число | Нет | Нет |
| Да | Brand\_id | Число | Нет | Нет |
| Да | Catrgory\_id | Число | Нет | Нет |
| да | Producer\_id | Число | Нет | Нет |

Таблица 2.5 – user\_role

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Пустые значения (Да/Нет) |
| Да | User\_id | Число | Нет | Нет |
| Да | Role\_id | Число | Нет | Нет |

Таблица 2.6 – category

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Пустые значения (Да/Нет) |
| PK | Id | Число | Нет | Нет |
| Нет | name | Строка | Нет | Нет |

Таблица 2.7 – brand

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Пустые значения (Да/Нет) |
| PK | id | Число | Нет | Нет |
| Нет | name | Строка | Нет | Нет |

Таблица 2.8 – role

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ (Да/Нет) | Поле | Тип | Значение по умолчанию | Пустые значения (Да/Нет) |
| PK | Id | Число | Нет | Нет |
| Нет | name | Строка | Нет | Нет |

В результате была спроектирована база данных для поддержки работы интернет магазина цифровой и бытовой техники. База данных включает в себя информацию о продуктах, списки брендов и категорий, информацию о пользователях и их ролях

## 2.3. Проектирование запросов к базе данных

Проектирование запросов на создание базы данных выглядит следующим образом [3]:

Листинг 1 – запрос на создание базы данных

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

-- -----------------------------------------------------

-- Schema mydb

-- -----------------------------------------------------

-- -----------------------------------------------------

-- Schema webstoredb

-- -----------------------------------------------------

DROP SCHEMA IF EXISTS `webstoredb` ;

-- -----------------------------------------------------

-- Schema webstoredb

-- -----------------------------------------------------

CREATE SCHEMA IF NOT EXISTS `webstoredb` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_0900\_ai\_ci ;

USE `webstoredb` ;

-- -----------------------------------------------------

-- Table `webstoredb`.`brand`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `webstoredb`.`brand` ;

CREATE TABLE IF NOT EXISTS `webstoredb`.`brand` (

`id` BIGINT NOT NULL,

`name` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `webstoredb`.`category`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `webstoredb`.`category` ;

CREATE TABLE IF NOT EXISTS `webstoredb`.`category` (

`id` BIGINT NOT NULL,

`name` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `webstoredb`.`my\_user`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `webstoredb`.`my\_user` ;

CREATE TABLE IF NOT EXISTS `webstoredb`.`my\_user` (

`id` BIGINT NOT NULL,

`address` VARCHAR(255) NULL DEFAULT NULL,

`fio` VARCHAR(255) NULL DEFAULT NULL,

`password` VARCHAR(255) NULL DEFAULT NULL,

`phone` VARCHAR(255) NULL DEFAULT NULL,

`username` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `UK\_sfp0l65piri344cgr5yiugcd3` (`username` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `webstoredb`.`producer`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `webstoredb`.`producer` ;

CREATE TABLE IF NOT EXISTS `webstoredb`.`producer` (

`id` BIGINT NOT NULL,

`address` VARCHAR(255) NULL DEFAULT NULL,

`phone` VARCHAR(255) NULL DEFAULT NULL,

`title` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `webstoredb`.`product`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `webstoredb`.`product` ;

CREATE TABLE IF NOT EXISTS `webstoredb`.`product` (

`id` BIGINT NOT NULL,

`description` VARCHAR(255) NULL DEFAULT NULL,

`filename` VARCHAR(255) NULL DEFAULT NULL,

`name` VARCHAR(255) NULL DEFAULT NULL,

`price` FLOAT NULL DEFAULT NULL,

`quantity` INT NULL DEFAULT NULL,

`brand\_id` BIGINT NULL DEFAULT NULL,

`category\_id` BIGINT NULL DEFAULT NULL,

`producer\_id` BIGINT NULL DEFAULT NULL,

PRIMARY KEY (`id`),

INDEX `FKs6cydsualtsrprvlf2bb3lcam` (`brand\_id` ASC) VISIBLE,

INDEX `FK1mtsbur82frn64de7balymq9s` (`category\_id` ASC) VISIBLE,

INDEX `FKaxeo9fj1sfah36yd9bujs8qft` (`producer\_id` ASC) VISIBLE,

CONSTRAINT `FK1mtsbur82frn64de7balymq9s`

FOREIGN KEY (`category\_id`)

REFERENCES `webstoredb`.`category` (`id`),

CONSTRAINT `FKaxeo9fj1sfah36yd9bujs8qft`

FOREIGN KEY (`producer\_id`)

REFERENCES `webstoredb`.`producer` (`id`),

CONSTRAINT `FKs6cydsualtsrprvlf2bb3lcam`

FOREIGN KEY (`brand\_id`)

REFERENCES `webstoredb`.`brand` (`id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `webstoredb`.`role`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `webstoredb`.`role` ;

CREATE TABLE IF NOT EXISTS `webstoredb`.`role` (

`id` BIGINT NOT NULL,

`name` VARCHAR(255) NULL DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE INDEX `UK\_8sewwnpamngi6b1dwaa88askk` (`name` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

-- -----------------------------------------------------

-- Table `webstoredb`.`user\_role`

-- -----------------------------------------------------

DROP TABLE IF EXISTS `webstoredb`.`user\_role` ;

CREATE TABLE IF NOT EXISTS `webstoredb`.`user\_role` (

`user\_id` BIGINT NOT NULL,

`role\_id` BIGINT NOT NULL,

UNIQUE INDEX `customers\_roles` (`user\_id` ASC, `role\_id` ASC) VISIBLE,

INDEX `FKa68196081fvovjhkek5m97n3y` (`role\_id` ASC) VISIBLE,

CONSTRAINT `FKa68196081fvovjhkek5m97n3y`

FOREIGN KEY (`role\_id`)

REFERENCES `webstoredb`.`role` (`id`),

CONSTRAINT `FKs7o7nqnu8qlbiq3g92rhjuvce`

FOREIGN KEY (`user\_id`)

REFERENCES `webstoredb`.`my\_user` (`id`))

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8mb4

COLLATE = utf8mb4\_0900\_ai\_ci;

## 2.4. Программная реализация и документирование базы данных

Разрабатываемое приложение сделано на основе MVC [4].

MVC расшифровывается как **Model—View—Controller**. Это принцип построения архитектуры большого приложения, при котором оно разбивается на три части.

Первая часть содержит всю бизнес-логику приложения. Такая часть называется Модель (Model). В ней содержится код, который делает все то, для чего приложение создавалось. Эта часть наиболее независимая от остальных.

Вторая часть содержит все, что касается отображения данных пользователю. Такая часть называется Вид (View). Именно в ней содержится код, который управляет показом окон, страниц, сообщений и т.д.

Третья часть содержит код, который занимается обработкой действий пользователя. Любые действия пользователя, направленные на изменения модели, должны обрабатываться тут. Такая часть называется Controller.

Такой подход позволяет независимо делать три вещи: логику программы (Model), механизм показа всех данных программы пользователю (View), обрабатывать ввод/действия пользователя (Controller).

Очень часто у приложения бывает несколько View. Это нормально. Вы можете видеть в Excel одни и те же данные в виде чисел и диаграмм. В играх вы можете смотреть на события от лица героя, на карте, со стороны и т.д. Все это – это разные View для одной Model.

В Controller’е сосредоточен весь код, который принимает решение, что менять в модели в ответ на действия пользователя. Например, пользователь принял решение закрыть программу, тогда надо сохранить данные модели в файл на диск. Или пользователь ввел новые данные. Тогда надо добавить их в модель, а модель потом уведомит все View об изменении данных, чтобы они отображали только актуальное их состояние.

— С точки зрения Java-программиста, можно сказать, что Model, View, Controller – это три группы классов, у которых:

* у каждой части есть свое назначение;
* связи между классами одной группы довольно сильные;
* связи между группами довольно слабые;
* способы взаимодействия частей довольно сильно регламентированы.

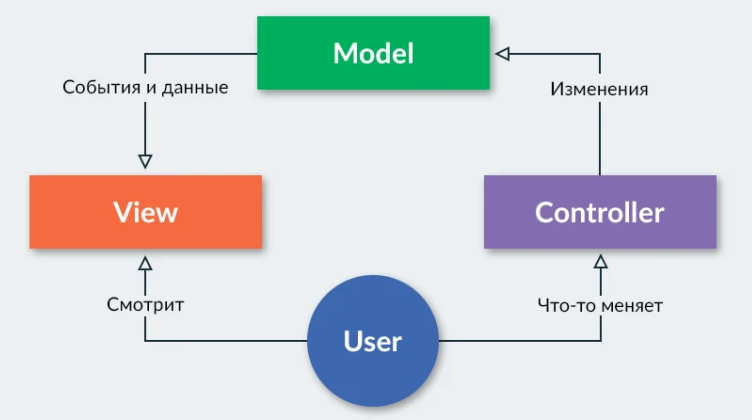


Рисунок 2.2 – Взаимодействие в MVC

Модель – это самая независимая часть системы. Она не зависит от View & Controller. Модель не может использовать классы из разделов View & Controller(!).

Основное ограничение вида (View) – вид не может менять модель. Классы вида могут обращаться к модели за данными или подписываться на события, но менять модель классы View не могут.

Основное ограничение контроллера – он не занимается отображением данных. Контроллер обрабатывает действия пользователя и меняет в соответствии с ними модель.

## 2.5. Проектирование разрабатываемого приложения

Приложение будет представлять из себя сайт, состоящий из страниц, каждая из которых будет отвечать за свои действия.

Так, на главной странице будет показан список продуктов при помощи следующего кода [5].

Листинг кода 2 – отображение списка продуктов

<section class=*"productList"*>

<div class=*"container"*>

<h3 class=*"text-center"*

th:text=*"${names}?${names}:#{messages.products}"*>Products</h3>

<table>

<thead>

<tr>

<th th:text=*"#{messages.category}"*>Category</th>

<th th:text=*"#{messages.brands}"*>Brand</th>

<th th:text=*"#{messages.producer}"*>Producer</th>

<th th:text=*"#{messages.name}"*>Name</th>

<th th:text=*"#{messages.price}"*>Price</th>

<th th:text=*"#{messages.quantity}"*>Quantity</th>

<th th:text=*"#{messages.description}"*>Description</th>

</tr>

</thead>

<tbody>

<tr th:each=*"product:${products}"*>

<td th:text=*"${product.category.name}"*></td>

<td th:text=*"${product.brand.name}"*></td>

<td th:text=*"${product.producer.title}"*></td>

<td th:text=*"${product.name}"*></td>

<td th:text=*"${product.price}"*></td>

<td th:text=*"${product.quantity}"*></td>

<td th:text=*"${product.description}"*></td>

</tr>

</tbody>

</table>

</div>

</section>

Поскольку нам необходимо ограничивать/разделять возможности менеджера и администратора, делать это будем не только посредством внутренней защиты на стороне сервера, но также и доступам к меню на стороне клиента.

Листинг кода 3 – ограничение к пунктам меню

<nav class=*"navbar navbar-expand-sm navbar-dark bg-primary border-bottom border-secondary"* >

<a class=*"navbar-brand nav-item "* href=*"/?language=ru"* >

<div>

<img src=*"/file/logo.png"* style="height: *36px*; width: *36px*">

<label th:text=*"#{messages.home}"*>Home</label>

</div>

</a>

<button class=*"navbar-toggler"* type=*"button"* data-toggle=*"collapse"* data-target=*"#navbarNavAltMarkup"* aria-controls=*"navbarNavAltMarkup"*

aria-expanded=*"false"* aria-label=*"Toggle navigation"*>

<span class=*"navbar-toggler-icon"*></span>

</button>

<div class=*"collapse navbar-collapse"* id=*"navbarNavAltMarkup"*>

<div class=*"navbar-nav"* th:if=*"${#request.userPrincipal != null}"*>

<li class=*"nav-item dropdown"*>

<a class=*"nav-link dropdown-toggle "* href=*"#"* id=*"navbarDropdown"* role=*"button"* data-toggle=*"dropdown"* aria-haspopup=*"true"* aria-expanded=*"false"* th:text=*"#{messages.categories}"*>

Categories

</a>

<div class=*"dropdown-menu"* aria-labelledby=*"navbarDropdown"*>

<a class=*"dropdown-item"* th:each=*"cat:${categories}"* th:href=*"'/cat'+${cat.id}"* th:text=*"${cat.name}"*>Action</a>

</div>

</li>

<li class=*"nav-item dropdown"*>

<a class=*"nav-link dropdown-toggle"* href=*"#"* id=*"navbarDropdown"* role=*"button"* data-toggle=*"dropdown"* aria-haspopup=*"true"* aria-expanded=*"false"* th:text=*"#{messages.brands}"*>

Brands

</a>

<div class=*"dropdown-menu"* aria-labelledby=*"navbarDropdown"*>

<a class=*"dropdown-item"* th:each=*"brand:${brands}"* th:href=*"'/bran'+${brand.id}"* th:text=*"${brand.name}"*>Another action</a>

</div>

</li>

<li class=*"nav-item dropdown"* th:if=*"${#request.isUserInRole('MANAGER') or #request.isUserInRole('ADMIN')}"*>

<a class=*"nav-link dropdown-toggle"* href=*"#"* id=*"navbarDropdown"* role=*"button"* data-toggle=*"dropdown"* aria-haspopup=*"true"* aria-expanded=*"false"* th:text=*"#{messages.lists}"*>

Lists

</a>

<div class=*"dropdown-menu"* aria-labelledby=*"navbarDropdown"*>

<a class=*"dropdown-item"* href=*"/category"* th:text=*"#{messages.categories}"*>Category</a>

<a class=*"dropdown-item"* href=*"/brand"* th:text=*"#{messages.brands}"*>Brand</a>

<a class=*"dropdown-item"* href=*"/product"* th:text=*"#{messages.products}"*>Products</a>

</div>

</li>

<li class=*"nav-item dropdown"*>

<a class=*"nav-link dropdown-toggle "*id=*"navbarDropdown"* role=*"button"* data-toggle=*"dropdown"* aria-haspopup=*"true"* aria-expanded=*"false"* th:text=*"#{messages.adminList}"* th:if=*"${#request.isUserInRole('ADMIN')}"*>

Admin list

</a>

<div class=*"dropdown-menu"* aria-labelledby=*"navbarDropdown"*>

<a class=*"dropdown-item"* href=*"/producer"* th:text=*"#{messages.producers}"*>Producer</a>

<a class=*"dropdown-item"* href=*"/manager"* th:text=*"#{messages.managers}"*>Managers</a>

<a class=*"dropdown-item"* href=*"/user"* th:text=*"#{messages.users}"*>Users</a>

</div>

</li>

<li class=*"nav-item dropdown"*>

<a class=*"nav-link dropdown-toggle"* href=*"#"* id=*"navbarDropdown"* role=*"button"* data-toggle=*"dropdown"* aria-haspopup=*"true"* aria-expanded=*"false"* th:text=*"#{messages.language}"*>

Language

</a>

<div class=*"dropdown-menu"* aria-labelledby=*"navbarDropdown"*>

<a class=*"dropdown-item"* th:href=*"@{|${#httpServletRequest.requestURI}?language=ru|}"* th:text=*"#{messages.russian}"*>RU</a>

<a class=*"dropdown-item"* th:href=*"@{|${#httpServletRequest.requestURI}?language=us|}"* th:text=*"#{messages.english}"*>US</a>

</div>

</li>

<a class=*"nav-item nav-link"* href=*"/logout?language=ru"* th:text=*"#{messages.logout}"*>LogOut</a>

</div>

<div class=*"navbar-nav"* th:if=*"${#request.userPrincipal == null}"*>

<a class=*"nav-item nav-link"* href=*"/login"* th:text=*"#{messages.login}"*>LogIn</a>

<a class=*"nav-item nav-link"* href=*"/registration"* th:text=*"#{messages.singin}"*>SingIn</a>

</div>

</div>

</nav>

**3. Применение разработанной базы данных**

**3.1. Руководство пользователя**

Главная страница веб-интерфейса для поддержки базы данных представлена на рисунке 3.1 - вид от менеджера, и 3.2 – вид от администратора.

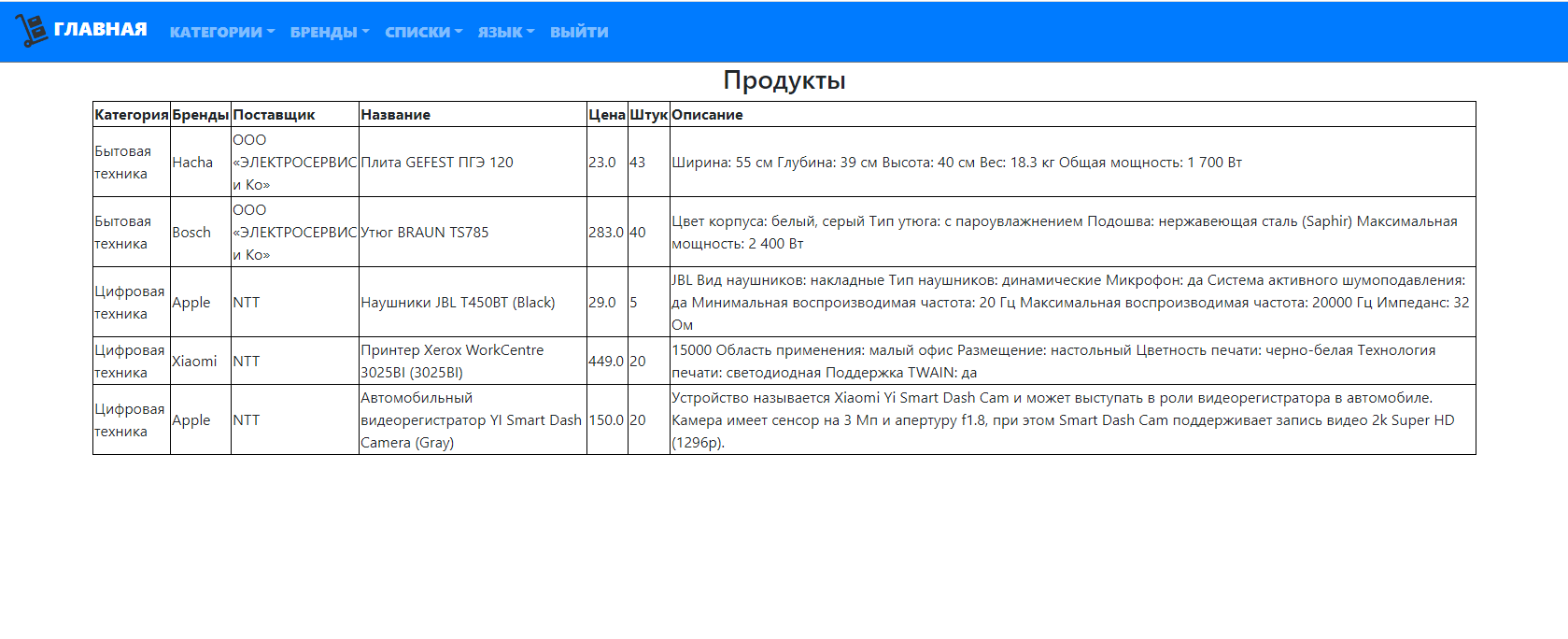


Рисунок 3.1 – Главная страница. Вид менеджера

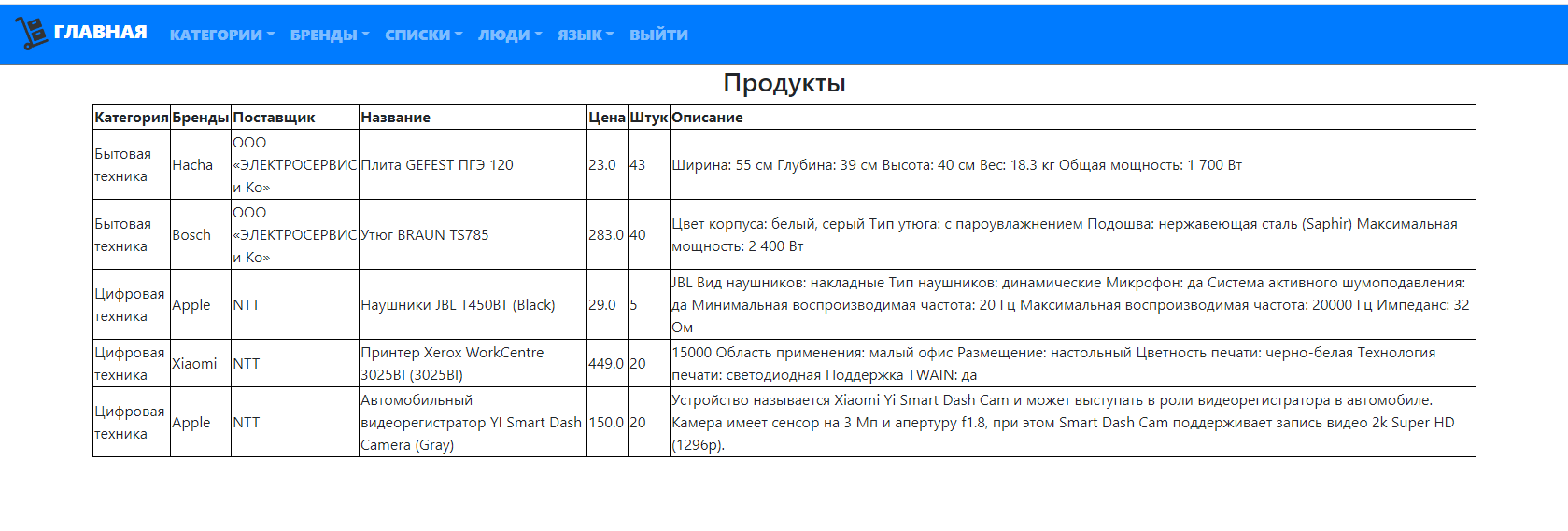


Рисунок 3.2 – Главная страница, вид Администратора

Для быстрого доступа к товаром определенной категории или определенного бренда, пользователю достаточно воспользоваться выпадающим меню, как показано на рисунках 3.3-3.4

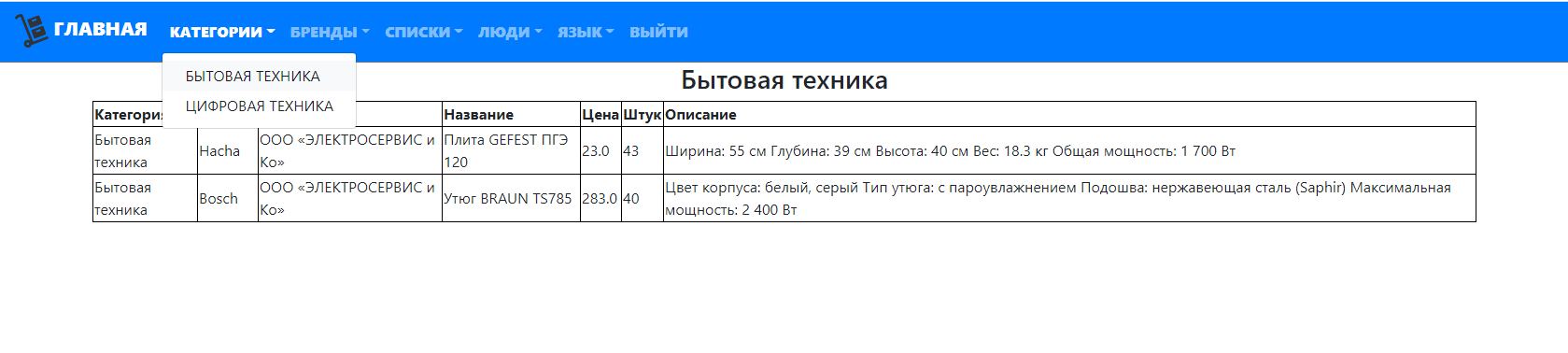


Рисунок 3.3 – Быстрый доступ к товарам по категории

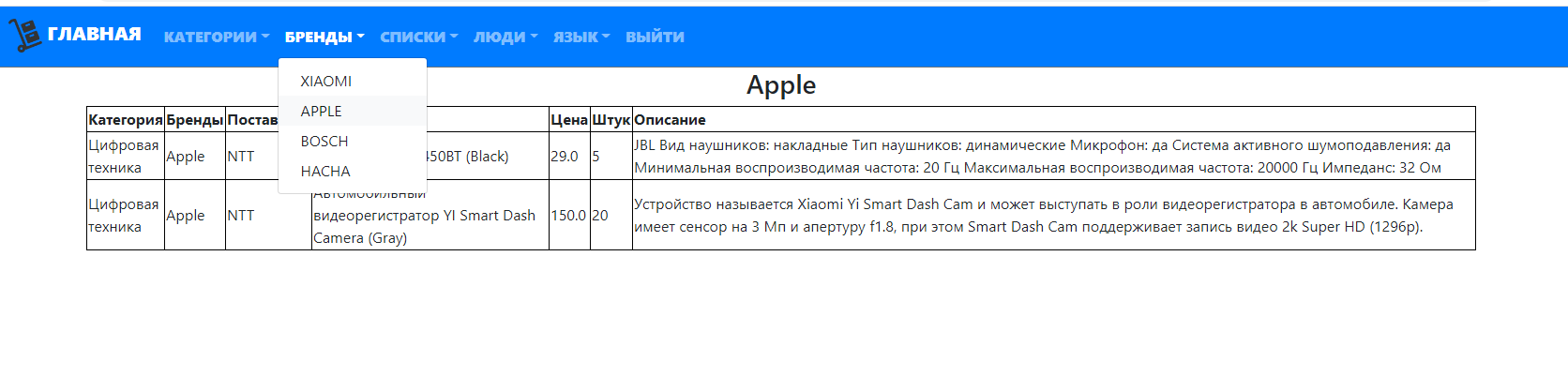


Рисунок 3.4 – Быстрый доступ к товарам по бренду

Для работы со с данными таблиц списков, достаточно выбрать нужный вариант таблицы из выпадающего меню «Списки»

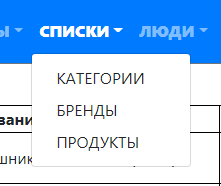


Рисунок 3.5 – меню выпадающего списка

Пример страницы категорий показан на рисунке ниже. Радиокнопка позволяет выбрать категорию, которая будет изменена при нажатии кнопки «Редактировать»

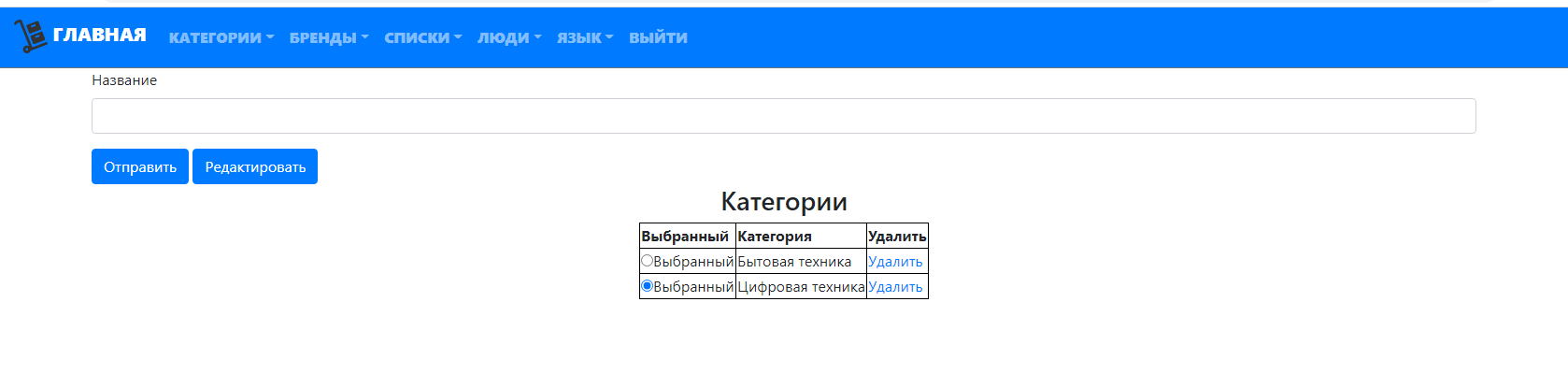


Рисунок 3.6 – страница работы с категориями

Страница работы с брендами выглядит похожим образом и имеет тот же функционал

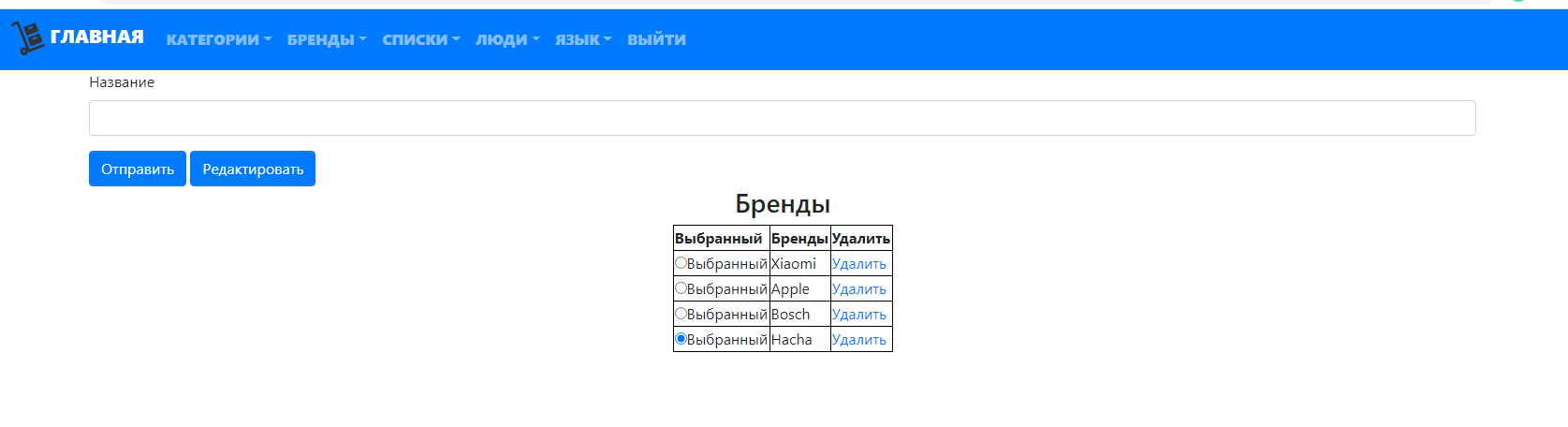


Рисунок 3.7 – страница работы с брендами

Страница работы с продуктами сделана тем же образом. При нажатии на радиокнопку, все поля заполнятся данными выбранного продукта, и можно будет редактировать его без проблем

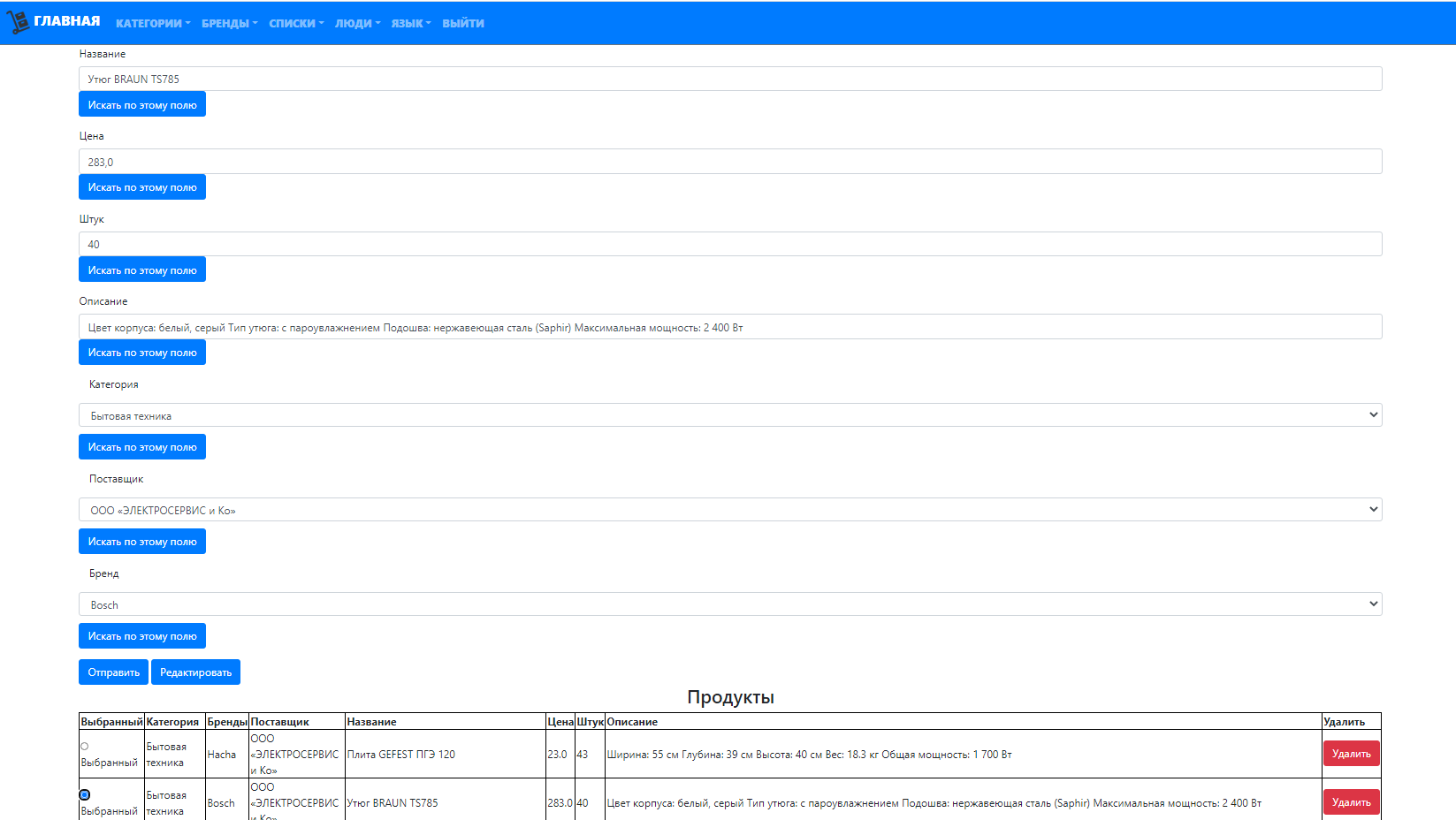


Рисунок 3.8 – страница работы с продуктами

Работать с таблицами, данные которые связаны с людьми моет только администратор. Список таблиц, с которыми он может работать представлен ниже

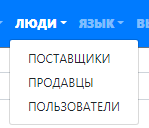


Рисунок 3.9 – меню-выпадающий список администратора

Страница управления пользователями выглядит следующим образом

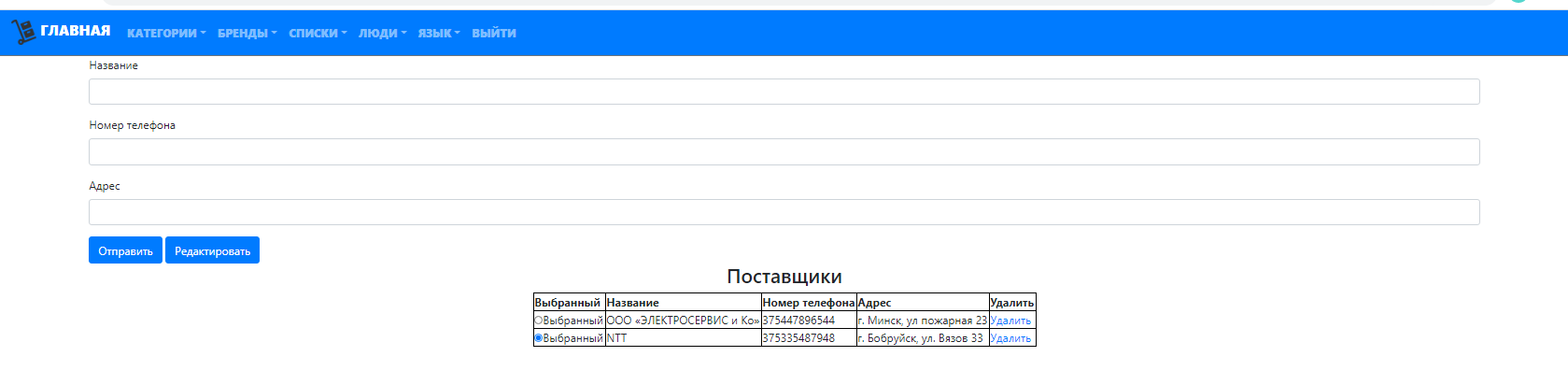


Рисунок 3.10 – страница управления пользователями

Тот же вид для управления менеджерами (продавцами) и пользователями. Пароль хранится в зашифрованном виде

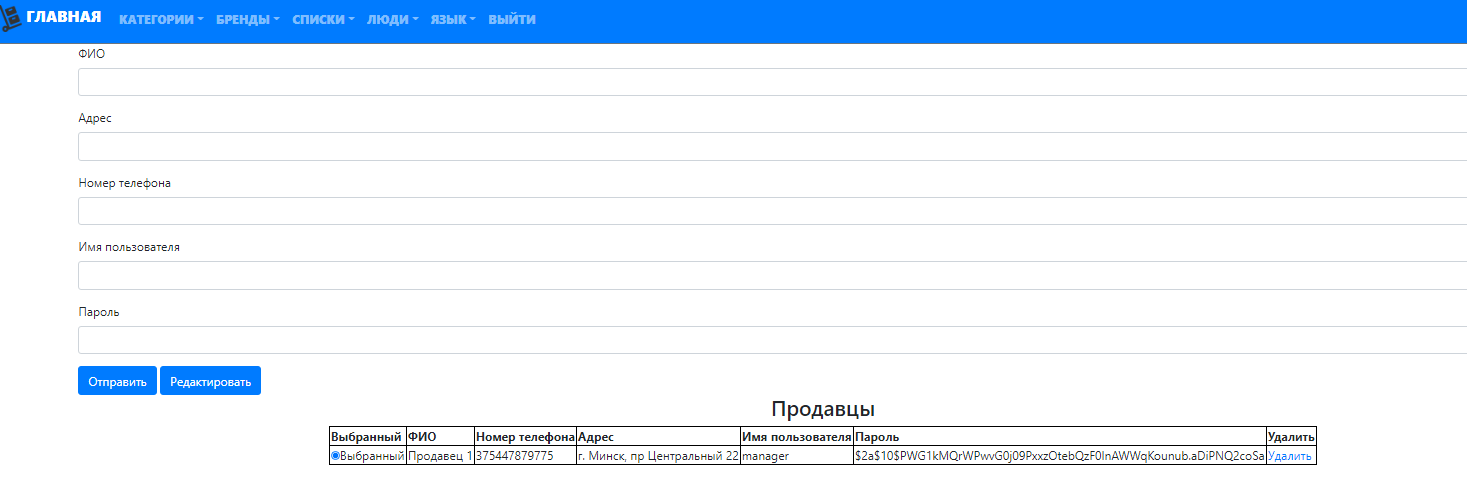


Рисунок 3.11 – Страница управления менеджерами

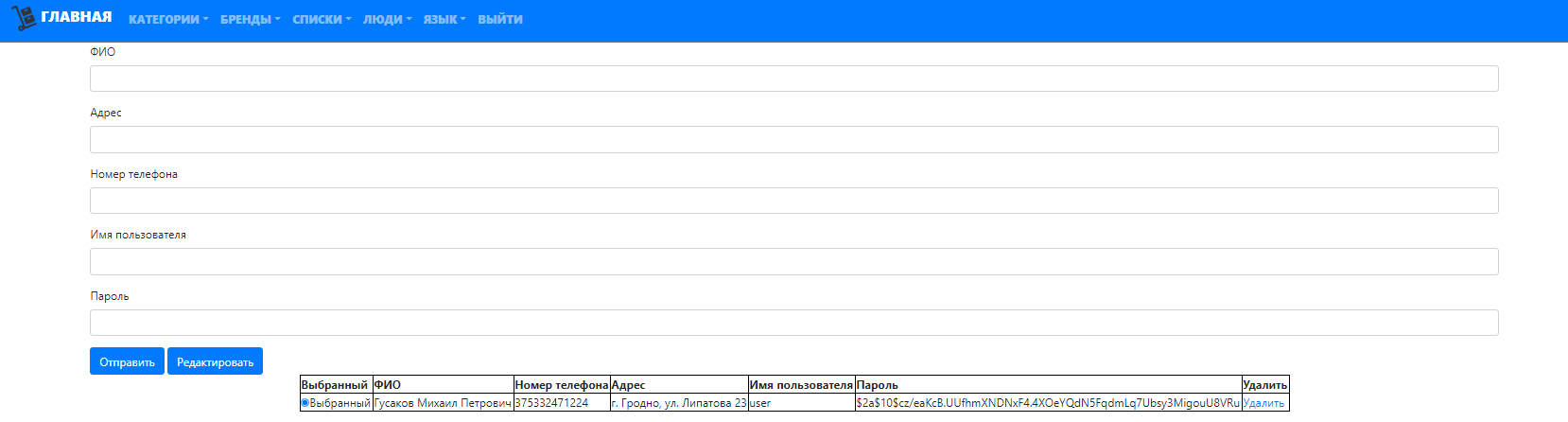


Рисунок 3.Страница управления пользователями

Для удобства, в зависимости от человека, который будет работать, сделано переключение языка

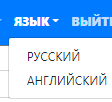


Рисунок 3.12 – меню переключения языка

## 3.2. Администрирование базы данных

Администрирование базы данных производится благодаря формам, описанным в пункте 3.1. После любой работы с данными (за исключением чтения), пользователю будет показано уведомление об успешном или провальном результате для каждой из таблиц.

К примеру, если мы попытаемся создать категорию, чье имя превышает все допустимые размеры, мы получим следующее уведомление об ошибке

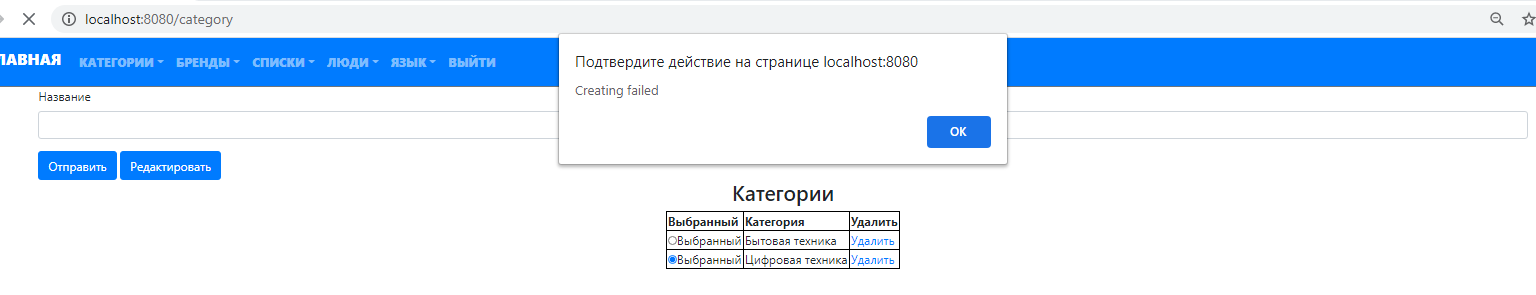


Рисунок 3.13 – уведомление об ошибке добавления записи

Также, если мы попробуем создать новую категорию, которая подходит по параметрам, к примеру «Новая категория», то получим уведомление об успешном создании

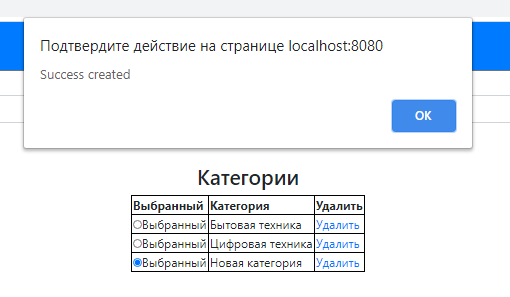


Рисунок 3.14 – уведомление об успешном добавлении записи

То же касается и удаления.

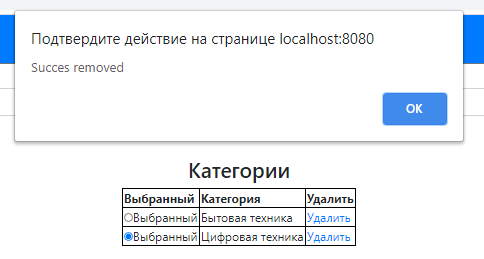


Рисунок 3.15 – уведомление об успешном удалении записи

## 3.3. Реализация клиентских запросов

Благодаря реализации приложения на MVC, все запросы страницы обрабатывает соответствующий контроллер [6]. К примеру, если мы будет отправлять данные на странице продуктов, все запросы будет обрабатывать ProductController.

Листинг кода 4 – Реализация контроллера продуктов для ответы на клиентские запросы

@Controller

@RequestMapping("/product")

**public** **class** ProductController {

@Autowired

ProductService service;

@Autowired

CategoryService categoryService;

@Autowired

ProducerService producerService;

@Autowired

BrandService brandService;

@Value("${upload.path}")

String path;

@Autowired

ListService listService;

@GetMapping

**public** String getproduct(Model model) {

listService.makeModel(model);

List<Product> findAll = service.repository.findAll();

**if** (!findAll.isEmpty())

model.addAttribute("products", findAll);

List<Producer> producer = producerService.repository.findAll();

**if** (!producer.isEmpty())

model.addAttribute("producers", producer);

**return** "product";

}

@PostMapping

**public** String setproduct(Model model, @ModelAttribute Product product, **int** categoryId, **int** producerId, **int** brandId,

RedirectAttributes ra) {

String response;

System.***out***.println("start create product");

**try** {

Category categ = categoryService.read(categoryId);

Producer producer = producerService.read(producerId);

Brand brand = brandService.read(brandId);

product.setCategory(categ);

product.setProducer(producer);

product.setBrand(brand);

service.create(product);

categ.getProduct().add(product);

producer.getProduct().add(product);

brand.getProduct().add(product);

categoryService.update(categ);

producerService.update(producer);

brandService.update(brand);

response = "Success created";

ra.addFlashAttribute("response1", "Product was added");

} **catch** (Exception e) {

e.printStackTrace();

response = "creating failed";

ra.addFlashAttribute("response1", "Adding product failed");

}

System.***out***.println("create product: " + response);

List<Product> findAll = service.repository.findAll();

**if** (!findAll.isEmpty())

model.addAttribute("products", findAll);

**return** "redirect:/product";

}

@PostMapping("/edit")

**public** String setproduct(Model model, @ModelAttribute Product product, **int** categoryId, **int** producerId, **int** brandId,

@Param("idd") **int** idd, RedirectAttributes ra) {

**try** {

Product read = service.read(idd);

read.setName(product.getName());

read.setDescription(product.getDescription());

read.setPrice(product.getPrice());

read.setQuantity(product.getQuantity());

Category categ = categoryService.read(categoryId);

Producer producer = producerService.read(producerId);

Brand brand = brandService.read(brandId);

read.setCategory(categ);

read.setProducer(producer);

read.setBrand(brand);

service.update(read);

ra.addFlashAttribute("response1", "Product was edited");

} **catch** (Exception e) {

e.printStackTrace();

ra.addFlashAttribute("response1", "Editing product dailed");

}

**return** "redirect:/product";

}

@GetMapping("/remove/{id}")

**public** String removeproduct(Model model, @PathVariable Long id, RedirectAttributes ra) {

**try** {

System.***out***.println(id);

service.repository.delete(service.read(id));

ra.addFlashAttribute("response1", "Product was deleted");

} **catch** (Exception e) {

ra.addFlashAttribute("response1", "Deleting product failed");

}

**return** "redirect:/product";

}

**public** **void** makeProducers(Model model) {

List<Producer> producer = producerService.repository.findAll();

**if** (!producer.isEmpty())

model.addAttribute("producers", producer);

}

@PostMapping("/findByName")

**public** String findByName(Model model, @Param("name") String name) {

listService.makeModel(model);

makeProducers(model);

**try** {

List<Product> list = service.repository.findByNameContaining(name);

**if** (!list.isEmpty())

model.addAttribute("products", list);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** "product";

}

@PostMapping("/findByPrice")

**public** String findByPrice(Model model, @RequestParam(name = "price", defaultValue = "0") **float** price) {

listService.makeModel(model);

makeProducers(model);

**try** {

List<Product> list = service.repository.findByPrice(price);

**if** (!list.isEmpty())

model.addAttribute("products", list);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** "product";

}

@PostMapping("/findByQuantity")

**public** String findByquantity(Model model, @RequestParam(name = "quantity", defaultValue = "0") **int** quantity) {

listService.makeModel(model);

makeProducers(model);

**try** {

List<Product> list = service.repository.findByQuantity(quantity);

**if** (!list.isEmpty())

model.addAttribute("products", list);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** "product";

}

@PostMapping("/findByDescription")

**public** String findByDescription(Model model, @RequestParam(name = "description") String description) {

listService.makeModel(model);

makeProducers(model);

**try** {

List<Product> list = service.repository.findByDescriptionContaining(description);

**if** (!list.isEmpty())

model.addAttribute("products", list);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** "product";

}

@PostMapping("/findByCategory")

**public** String findByCategory(Model model, @RequestParam(name = "categoryId") **int** categoryId) {

listService.makeModel(model);

makeProducers(model);

**try** {

List<Product> list = service.repository.findByCategory(categoryService.read(categoryId));

**if** (!list.isEmpty())

model.addAttribute("products", list);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** "product";

}

@PostMapping("/findByProducer")

**public** String findByProducer(Model model, @RequestParam(name = "producerId") **int** producerId) {

listService.makeModel(model);

makeProducers(model);

**try** {

List<Product> list = service.repository.findByProducer(producerService.read(producerId));

**if** (!list.isEmpty())

model.addAttribute("products", list);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** "product";

}

@PostMapping("/findByBrand")

**public** String findByBrand(Model model, @RequestParam(name = "brandId") **int** brandId) {

listService.makeModel(model);

makeProducers(model);

**try** {

List<Product> list = service.repository.findByBrand(brandService.read(brandId));

**if** (!list.isEmpty())

model.addAttribute("products", list);

} **catch** (Exception e) {

e.printStackTrace();

}

**return** "product";

}

}

Все взаимодействие с базой данных между пользователем и базой берет на себя Spring с его магическими методами. Программисту остается лишь создать сервис с необходимыми операциями для каждой сущности

Листинг кода 6 – Реализация взаимодействия сущности с базой данных

**package** com.group.webstorebase.service;

**import** org.springframework.data.jpa.repository.JpaRepository;

**public** **class** CrudImpl<Entity> **implements** CrudService<Entity> {

**public** JpaRepository<Entity, Long> repository;

@Override

**public** **void** create(Entity entity) **throws** Exception {

repository.saveAndFlush(entity);

}

@Override

**public** Entity read(**long** id) {

**return** repository.findById(id).isPresent() ? repository.findById(id).get() : **null**;

}

@Override

**public** **void** update(Entity entity) **throws** Exception {

repository.saveAndFlush(entity);

}

@Override

**public** **void** delete(**long** id) **throws** Exception {

repository.deleteById(id);

}

**public** CrudImpl(JpaRepository<Entity, Long> repository) {

**super**();

**this**.repository = repository;

}

}

## 3.4. Обоснование и реализация механизма обеспечения безопасности и сохранности данных

Spring Security – это платформа для приложений Java и Java EE, которая предоставляет специализированные механизмы для построения систем авторизации и аутентификации [7]. Она представляет собой настраиваемую систему для проверки подлинности и контроля доступа к определенным ресурсам приложений Java и Java EE, а также защиты корпоративных приложений, созданных с помощью Spring Framework (см. рисунок 1). Впервые разработку данной платформы начал Бен Алекс (Ben Alex) в 2003 году, называлась она «Acegi Security», после этого в 2004 году был выпущен ее первый релиз.Однако долго самостоятельным проектом она не просуществовала и была поглощена компанией , как следствие этого она стала его официальным дочерним проектом. Под юрисдикцией компании «Spring» платформа была публично представлена под новым именем «Spring Security» версии релиза 2.0.0 в апреле 2008 года.

Основные объекты контекста Spring Security

SecurityContextHolder

В этом ключевом объекте хранится информация о текущем контексте безопасности определенного приложения, включая частную информацию о самом пользователе, который работает в настоящем времени с приложением. Этот объект по умолчанию использует ThreadLocal, для того чтобы хранить эти сведений, что означает, что контекст безопасности всегда доступен для методов в том же потоке выполнения, даже если контекст безопасности явно не передается в качестве аргумента этим методам, однако использование его будет безопасным, при соблюдении и принятии мер для очищения потока после завершения обработки запроса текущего пользователя. Несмотря на всю универсальность этого метода, зачастую в силу специфики работы самого метода с потоками, его использование не является хорошим решением.

Authentication

Внутри SecurityContext, содержится объект Authentication. Он используется в ситуациях, когда есть необходимость необходимость в информации системы безопасности, которая связана с запросами пользователя, то есть Spring Security использует этот объект для предоставления данной информации. По обыкновению, нет особой необходимости в создании объекта Authentication своими силами, однако запросы к нему довольно частое явление.

UserDetails

Сервис UserDetails представляет собой принципала (пользователя), но в более расширенном виде и с учетом специфики используемого приложения. Единственный метод этого сервиса принимает имя пользователя в виде String и возвращает UserDetails, представляя собой общий подход к загрузке информации о пользователе в «Spring Security»

В случае успешной аутентификации, userDetail используется для создания Authentification объекта, который хранится в SecurityContextHolder.

GrantedAuthority

Еще одним ключевым методом является getAuthorities(), который представляет собой большой массив объектов GrantedAuthority. GrantedAuthority - это определенные полномочия, которые предоставляются принципалу, например, ROLE\_ADMINISTRATOR или ROLE\_HR\_SUPERVISOR, которые позже настраиваются для веб-авторизации, а также авторизации методов и авторизации объектов домена. Область видимости для разрешений, предоставляемых GrantedAuthority, является все приложение, а не отдельная предметная область.

Пример аутентификации в веб-приложениях

Самая обыкновенная ситуация, когда пользователь сталкивается с аутентификацией в веб-приложении:

1. Вы заходите на главную страницу и нажимаете на ссылку.
2. Запрос идет на сервер и сервер решает что вы запрашиваете защищенный ресурс.
3. Так как подлинность пользователя в настоящее время не подтверждена, то сервер отправляет ответ о том, что вы должны пройти процедуру аутентификации. Ответом будет либо код HTTP, либо перенаправление на особую веб-страницу.
4. В зависимости от механизма аутентификации, браузер либо перенаправит вас на специальную веб-страницу, где можно будет заполнить форму, либо браузер каким-то другим образом получит данные подтверждающие подлинность вашей личности.
5. Браузер отошлет ответ на сервер. Это будет либо HTTP POST, с содержимым формы, которую вы заполнили, либо HTTP заголовок, содержащий ваши данные аутентификации.
6. Далее сервер будет решать, является ли «удостоверение личности» действительными. Если оно действительно, то будет выполнен следующий шаг. Если же оно недействительно, то как правило браузер предложит попробовать идентифицироваться еще раз.
7. Оригинальный запрос, который привел к запуску процедуры аутентификации, будет повторен. Надеемся, что вы имеете достаточно полномочий для доступа к защищенному ресурсу. Если вы имеете соответствующие права доступа, то запрос будет успешным. В противном случае вам будет возвращен код ошибки 403, что означает "запрещено".

Spring Security имеет явно реализованные классы отвечающие за большинство шагов, описанных выше. Основные участники (в порядке их использования) ExceptionTranslationFilter, AuthenticationEntryPoint и "механизм аутентификации", который отвечает за вызов AuthenticationManager.

Используя описанную технологию обеспечивается доступ к данным. Так, даже если пользователь знает адреса страниц для работы с таблицей поставщиков, но при этом не имеет роли администратора, то он не сможет что-то изменить даже если напрямую пошлет запрос.

В данном случае была рассмотрена база для системы управления доступом на основе использования ролей. Достаточно простой код позволяет построить гибкую масштабируемую систему разграничения прав пользователей. Однако должно внимание надо уделять управлению пользовательскими полномочиями на этапе использования, иначе гибкость может привести к хаосу и неразберихе.

# Заключение

Результатом выполнения курсового проекта является разработанная база данных для поддержки работы интернет-магазина.

В ходе проектирования веб-приложения были рассмотрены актуальные вопросы разработки и создания веб интерфейса базы данных. При этом, были решены следующие задачи

* проведен анализ предметной области
* спроектированная база данных была приведена к нормальной форме
* разработан веб-интерфейс для доступа к базе данных
* разграничены возможности работы с базой данных по ролям
* добавлена поддержка работы с каталогами товаров
* изучена возможность делать подобные проекты

В результате проведенных работ на базе выбранных технологий была создана база данных для поддержки работы интернет-магазина бытовых и цифровых товаров

# Список использованных источников

1. Базы данных [Электронный ресурс]. – Режим доступа: https://edu.tusur.ru/publications/763. – Дата доступа: 20.10.2020.
2. Организация интернет-магазина [Электронный ресурс]. – Режим доступа: https://studbooks.net/1193264/marketing/organizatsiya\_raboty\_internet\_magazina\_pravovoe\_regulirovanie. – Дата доступа: 20.10.2020.
3. Основы работы с MySql [Электронный ресурс]. – Режим доступа: http://mithrandir.ru/professional/soft-and-hardware/mysql-workbench-server.html. – Дата доступа: 20.10.2020.
4. Паттерн MVC [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/150267/. – Дата доступа: 20.10.2020.
5. Thymeleaf [Электронный ресурс]. – Режим доступа: https://www.thymeleaf.org/. – Дата доступа: 20.10.2020.
6. Spring Controller [Электронный ресурс]. – Режим доступа: https://www.codeflow.site/ru/article/spring-controller-vs-restcontroller. – Дата доступа: 20.10.2020.
7. Spring Security [Электронный ресурс]. – Режим доступа: https://ru.bmstu.wiki/Spring\_Securit. – Дата доступа: 20.10.2020.

# Приложение А – Код страницы работы с товарами

<!DOCTYPE html>

<html xmlns:th=*"http://www.thymeleaf.org"*>

<head>

<title th:text=*"#{messages.product}"*>Product</title>

<div th:replace=*"~{commons :: head}"*></div>

</head>

<body onload="onLoad()">

<div th:replace=*"~{commons :: nav}"*></div>

<form action=*"/product"* method=*"post"* enctype=*"multipart/form-data"*>

<section class=*"productForm container"*>

<div class=*"row"*>

<div class=*"col"*>

<div class=*"form-group"*>

<label th:text=*"#{messages.name}"*>Name</label> <input id=*"name"*

type=*"text"* name=*"name"* class=*"form-control"* required>

<button type=*"submit"* formaction=*"/product/findByName"*

formnovalidate class=*"btn btn-primary"*

th:text=*"#{messages.findByThisField}"*>Find</button>

</div>

<div class=*"form-group"*>

<label th:text=*"#{messages.price}"*>Price</label> <input id=*"price"*

type=*"number"* name=*"price"* class=*"form-control"* required>

<button type=*"submit"* formaction=*"/product/findByPrice"*

formnovalidate class=*"btn btn-primary"*

th:text=*"#{messages.findByThisField}"*>Find</button>

</div>

<div class=*"form-group"*>

<label th:text=*"#{messages.quantity}"*>Quantity:</label> <input

id=*"quantity"* type=*"number"* name=*"quantity"* class=*"form-control"*

required>

<button type=*"submit"* formaction=*"/product/findByQuantity"*

formnovalidate class=*"btn btn-primary"*

th:text=*"#{messages.findByThisField}"*>Find</button>

</div>

<div class=*"form-group"*>

<label th:text=*"#{messages.description}"*>Description::</label> <input

id=*"description"* type=*"text"* name=*"description"*

class=*"form-control"* required>

<button type=*"submit"* formaction=*"/product/findByDescription"*

formnovalidate class=*"btn btn-primary"*

th:text=*"#{messages.findByThisField}"*>Find</button>

</div>

<div class=*"form-group required"*>

<label class=*"col-sm-2 control-label"*

th:text=*"#{messages.category}"*>Category</label> <select

id=*"catName"* name=*"categoryId"* size=*"1"* class=*"form-control"*

required>

<option th:each=*"categ:${categories}"* th:value=*"${categ.id}"*

th:text=*"${categ.name}"*>categ</option>

</select>

<button type=*"submit"* formaction=*"/product/findByCategory"*

formnovalidate class=*"btn btn-primary"*

th:text=*"#{messages.findByThisField}"*>Find</button>

</div>

<div class=*"form-group required"*>

<label class=*"col-sm-2 control-label"*

th:text=*"#{messages.producer}"*>Producer</label> <select

id=*"prodName"* name=*"producerId"* size=*"1"* class=*"form-control"*

required>

<option th:each=*"producer:${producers}"* th:value=*"${producer.id}"*

th:text=*"${producer.title}"*>producer</option>

</select>

<button type=*"submit"* formaction=*"/product/findByProducer"*

formnovalidate class=*"btn btn-primary"*

th:text=*"#{messages.findByThisField}"*>Find</button>

</div>

<div class=*"form-group required"*>

<label class=*"col-sm-2 control-label"* th:text=*"#{messages.brand}"*>Brand</label>

<select id=*"brandName"* name=*"brandId"* size=*"1"*

class=*"form-control"* required>

<option th:each=*"brand:${brands}"* th:value=*"${brand.id}"*

th:text=*"${brand.name}"*>brand</option>

</select>

<button type=*"submit"* formaction=*"/product/findByBrand"*

formnovalidate class=*"btn btn-primary"*

th:text=*"#{messages.findByThisField}"*>Find</button>

</div>

<button type=*"submit"* class=*"btn btn-primary"*

th:text=*"#{messages.submit}"*>Submit</button>

<button th:if=*"${products}"* type=*"submit"*

formaction=*"/product/edit"* class=*"btn btn-primary"*

th:text=*"#{messages.edit}"*>Edit</button>

</div>

</div>

</section>

<section class=*"productList mb-5"*>

<div class=*"container"*>

<h3 class=*"text-center"* th:text=*"#{messages.products}"*>Products</h3>

<table>

<thead>

<tr>

<th th:text=*"#{messages.selectForEdit}"*>Category</th>

<th th:text=*"#{messages.category}"*>Category</th>

<th th:text=*"#{messages.brands}"*>Brand</th>

<th th:text=*"#{messages.producer}"*>Producer</th>

<th th:text=*"#{messages.name}"*>Name</th>

<th th:text=*"#{messages.price}"*>Price</th>

<th th:text=*"#{messages.quantity}"*>Quantity</th>

<th th:text=*"#{messages.description}"*>Description</th>

<th th:text=*"#{messages.remove}"*>Remove</th>

</tr>

</thead>

<tbody>

<tr th:each=*"product:${products}"* th:id=*"'id'+${product.id}"* >

<td>

<input th:onclick=*"'make('+${product.id}+')'"* type=*"radio"* name=*"idd"* th:value=*${product.id}* th:text=*"#{messages.selectForEdit}"* checked=*"checked"*>

</td>

<td th:id=*"catName"* th:text=*"${product.category.name}"*></td>

<td th:id=*"brandName"* th:text=*"${product.brand.name}"*></td>

<td th:id=*"prodName"* th:text=*"${product.producer.title}"*></td>

<td th:id=*"name"* th:text=*"${product.name}"*></td>

<td th:id=*"price"* th:text=*"${product.price}"*></td>

<td th:id=*"quantity"* th:text=*"${product.quantity}"*></td>

<td th:id=*"description"* th:text=*"${product.description}"*></td>

<td>

<a class=*"btn btn-danger"* th:href=*"'product/remove/'+${product.id}"* th:text=*"#{messages.remove}"*>Remove</a>

</td>

</tr>

</tbody>

</table>

</div>

</section>

<script type=*"text/javascript"*>

**function** make(id) {

$(".col #name").val(

$("section.productList tr#id" + id + " #name").html());

$(".col #price").val(

$("section.productList tr#id" + id + " #price").html());

$(".col #quantity").val(

$("section.productList tr#id" + id + " #quantity").html());

$(".col #description").val(

$("section.productList tr#id" + id + " #description").html());

$(

'.col #catName option:contains("'

+ $("section.productList tr#id" + id + " #catName")

.html() + '")').prop('selected', **true**);

$(

'.col #prodName option:contains("'

+ $("section.productList tr#id" + id + " #prodName")

.html() + '")').prop('selected', **true**);

$(

'.col #brandName option:contains("'

+ $("section.productList tr#id" + id + " #brandName")

.html() + '")').prop('selected', **true**);

}

</script>

</form>

</body>

</html>

# Приложение Б – Код обработки запросов сущности пользователь

package com.group.webstorebase.controller;

import java.util.ArrayList;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.data.repository.query.Param;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import com.group.webstorebase.entity.MyUser;

import com.group.webstorebase.entity.Role;

import com.group.webstorebase.service.ListService;

import com.group.webstorebase.service.MyUserService;

import com.group.webstorebase.service.RoleService;

@Controller

@RequestMapping("/user")

public class UserController {

@Autowired

MyUserService service;

@Autowired

RoleService roleService;

@Autowired

ListService listService;

@Autowired

private PasswordEncoder passwordEncoder;

@GetMapping

public String geUser(Model model) {

listService.makeModel(model);

Role roleU = roleService.findByName("ROLE\_USER");

Role roleM = roleService.findByName("ROLE\_MANAGER");

List<MyUser> findAll = service.repository.findAll();

if (!findAll.isEmpty()) {

List<MyUser> list = new ArrayList<MyUser>();

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

if(findAll.get(i).getRoles().contains(roleU) && !findAll.get(i).getRoles().contains(roleM))

list.add(findAll.get(i));

model.addAttribute("managers", list);

}

return "user";

}

@PostMapping

public String setUser(Model model, @ModelAttribute MyUser manager, RedirectAttributes ra) {

String response="error";

try {

service.register(manager);

ra.addFlashAttribute("response1","User was added");

} catch (Exception e) {

e.printStackTrace();

ra.addFlashAttribute("response1","Adding user failed");

}

return "redirect:/user";

}

@PostMapping("/edit")

public String editManager(@ModelAttribute MyUser manager, @Param("idd") int idd, RedirectAttributes ra) {

try {

MyUser read = service.read(idd);

read.setAddress(manager.getAddress());

read.setFio(manager.getFio());

read.setPassword(passwordEncoder.encode(manager.getPassword()));

read.setPhone(manager.getPhone());

read.setUsername(manager.getUsername());

service.update(read);

ra.addFlashAttribute("response1","User was edited");

} catch (Exception e) {

e.printStackTrace();

ra.addFlashAttribute("response1","Editing user failed");

}

return "redirect:/user";

}

@GetMapping("/remove/{id}")

public String removeManager(Model model, @PathVariable Long id, RedirectAttributes ra) {

try {

System.out.println(id);

service.repository.delete(service.read(id));

ra.addFlashAttribute("response1","User was deleted");}

catch (Exception e) {

ra.addFlashAttribute("response1","Deleting user failed");

}

return "redirect:/user";

}

}