1. ПРОЕКТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ
   1. Разработка структуры приложения.

В АРХИТЕКТУРУ

Данное приложение имеет архитектуру “клиент-сервер” с использованием MVC (Model-View-Controller) (Рисунок 3.1)

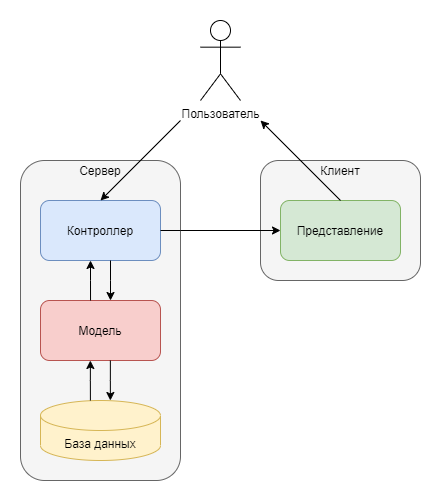


Рисунок 3.1 – Архитектура MVC

База данных хранит и исходные данные заполняемые администратором и данные пользователей и их заказы. Запросы к базе данных формируются в контроллерах. После получения данных, контроллер передает данную модель представлению на клиентскую сторону. Клиент из браузера может передавать определенные данные в контролеры на обработку и сохранение в базе данных.

Бизнес логика приложения находится в методах моделей (сущностей) и осуществляется в контроллерах. В представлениях на стороне пользователя нет бизнес логики.

Основное преимущество архитектуры MVC – модульность. Из-за этого свойства очень просто создавать множество представлений для одинаковых моделей (например представление модели “Категории” для администратора и пользователя). Так же с данной архитектурой намного проще отлаживать код (если, например, возникла ошибка в обработке данных, то сразу отбрасывается модель и представление, и ошибка ищется в контроллере).

* 1. Разработка алгоритмов обработки информации

Основные нетривиальные манипуляции с информацией – это функции связанные с характеристиками товаров и фильтрация по ним.

В АРХИТЕКТУРУ но и тут тоже! ПРО JSON описать подробнее json !

(ДОБАВИТЬ ПРО ФИЛЬТРАЦИЮ)

Основная преимущество хранения характеристик в виде JSON поля – возможность создавать и редактировать неограниченное количество характеристик, привязанных конкретно к одной категории, без создания дополнительных сущностей для характеристик. Из-за этого появляется другая проблема – обработка и создание этих характеристик.

Имеется класс CategoryCharacteristics – он находится в модели категории (сущность Category. Не записывается в базу данных), в виде List< CategoryCharacteristics> - так как может существовать много характеристик. Содержит поля string charactName (название самой характеристики) и класс CategoryCharacteristicsBool\_Value (объединяющий значения характеристики и значение, отвечающее за то, что являются ли эти значения числовыми и можно ли использовать для них фильтрация по диапазону) поля: List<string> charactValues, bool isNumeric. Данная странная на первый взгляд структура позволяет весьма просто сериализовать класс CategoryCharacteristics в поле типа JSON. При помощи словаря, вида Dictionary<string, CategoryCharacteristicsBool\_Value>, в который вносятся на место ключа название характеристик. После этого буквально одной встроенной функцией JsonConvert.SerializeObject из библиотеки Newtonsoft.Json производится сериализация данного обьекта в JSON. Далее данное значение сохраняется в базе данных в виде поля nvarchar(max); Десериализация происходит в обратном порядке с помощью функции JsonConvert.DeserializeObject.

Имеется класс ItemCharacteristics – он полностью зависит от класса рассмотренного выше (CategoryCharacteristics). Данный класс находится в модели товара (сущность Item. Не записывается в базу данных) в виде List< ItemCharacteristics> - так как может существовать много характеристик. Имеет поле string charactItemName – для имени характеристики, и поле string charactItemValue для значения характеристики. При создании нового товара (объекта Item) администратором – ему выдается список характеристик с их значениями из рассмотренного класса CategoryCharacteristics, после он выбирает для каждой характеристики одно значение (т.к. у товара может быть, например, только одна диагональ или один цвет). Далее название характеристики и ее значение записывается в объект ItemCharacteristics. При записи в базу данных с помощью объекта словаря вида Dictionary<string,string> записывается характеристика и ее значение, далее с помощью той же функции JsonConvert.SerializeObject происходит сериализация в JSON. Аналогично производится десериализация.

Фильтрация по описанным выше характеристикам происходит в классе FullFilter и Filter. Класс Filter имеет поля List<bool> exactValue, List<string> exactValueString, double? from и double? to. Поле фильтра exactValue – список значений bool, так как к каждому значению характеристики создается checkbox. При выборе значений характеристики пользователь меняет значения в этом списке с 0 на 1. Поля from и double нужны, если характеристика представляется диапазоном значений. Класс FullFilter включает в себя лист Filter – это и есть характеристики категории, также он включает схожие поля, описанные выше, только для цены и производителя (так как эти характеристики свойственны любому товару). Пользователь, применив фильтр, совершает POST запрос на сервер, передавая класс FullFilter. Он обрабатывается, находя те характеристики, значения которых были затронуты (или exactValue – есть хотя бы одно значение равное единице, или заполнены to и from). После этого формируется URL для GET запроса (например: … /filter?price=10000;15000&Разрешение%20экрана=HD;FullHD). Это нужно для того, чтобы пользователь мог обновить страницу и не отправлять повторный POST запрос, плюс это дает пользователю возможность сохранить фильтрацию (скопировав полный URL), если потребуется. Далее, совершается GET запрос, происходит обработка URL. URL “разбирается” и сопоставляется модели фильтра. После этого происходит фильтрация по известным значениям. Из-за такой архитектуры хранения характеристик приходится сначала получать весь список товаров категории, в которой происходит фильтрация, а далее, уже выбирать те товары, характеристики которых удовлетворяют полученным фильтрам. Сформированный список товаров, вместе с фильтром (для дальнейшей фильтрации) поступает в представление пользователя.

* 1. Логическая схема базы данных

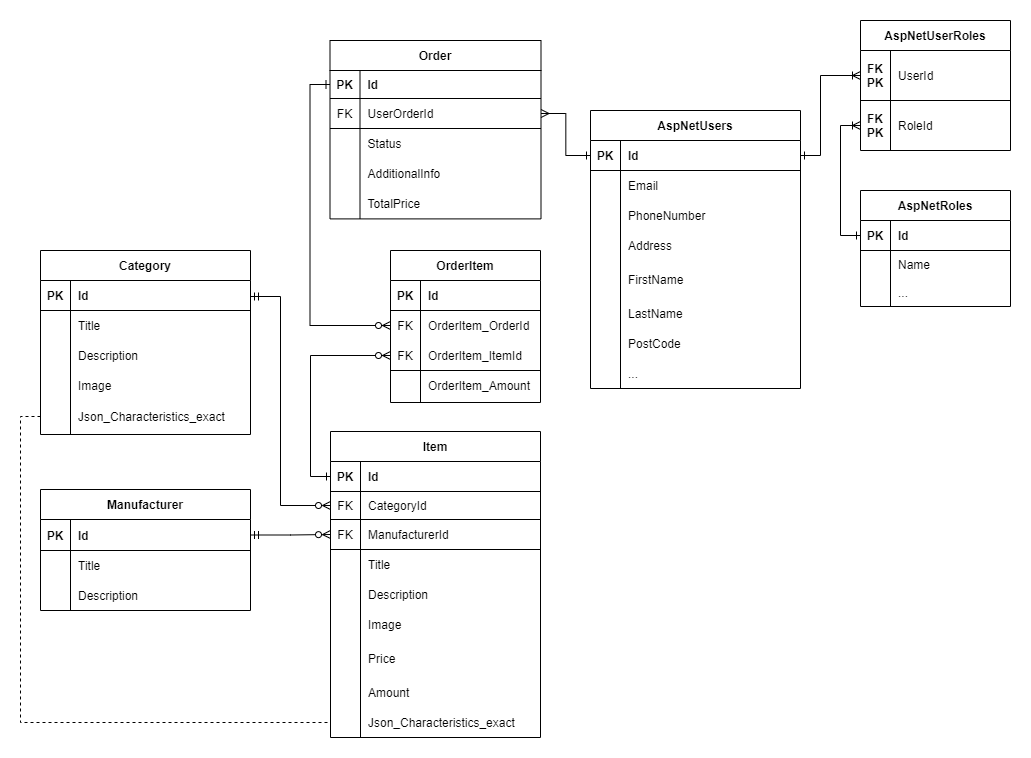


Рисунок 3.3 – Логическая схема базы данных

В АРХИТЕКТУРУ

* 1. Разработка интерфейса взаимодействия пользователя с системой

Были разработаны макеты веб-страниц приложения:

* Страница для просмотра категорий (домашняя страница) (Рисунок 3.4.1). При добавлении новых категорий страница будет более заполненной.

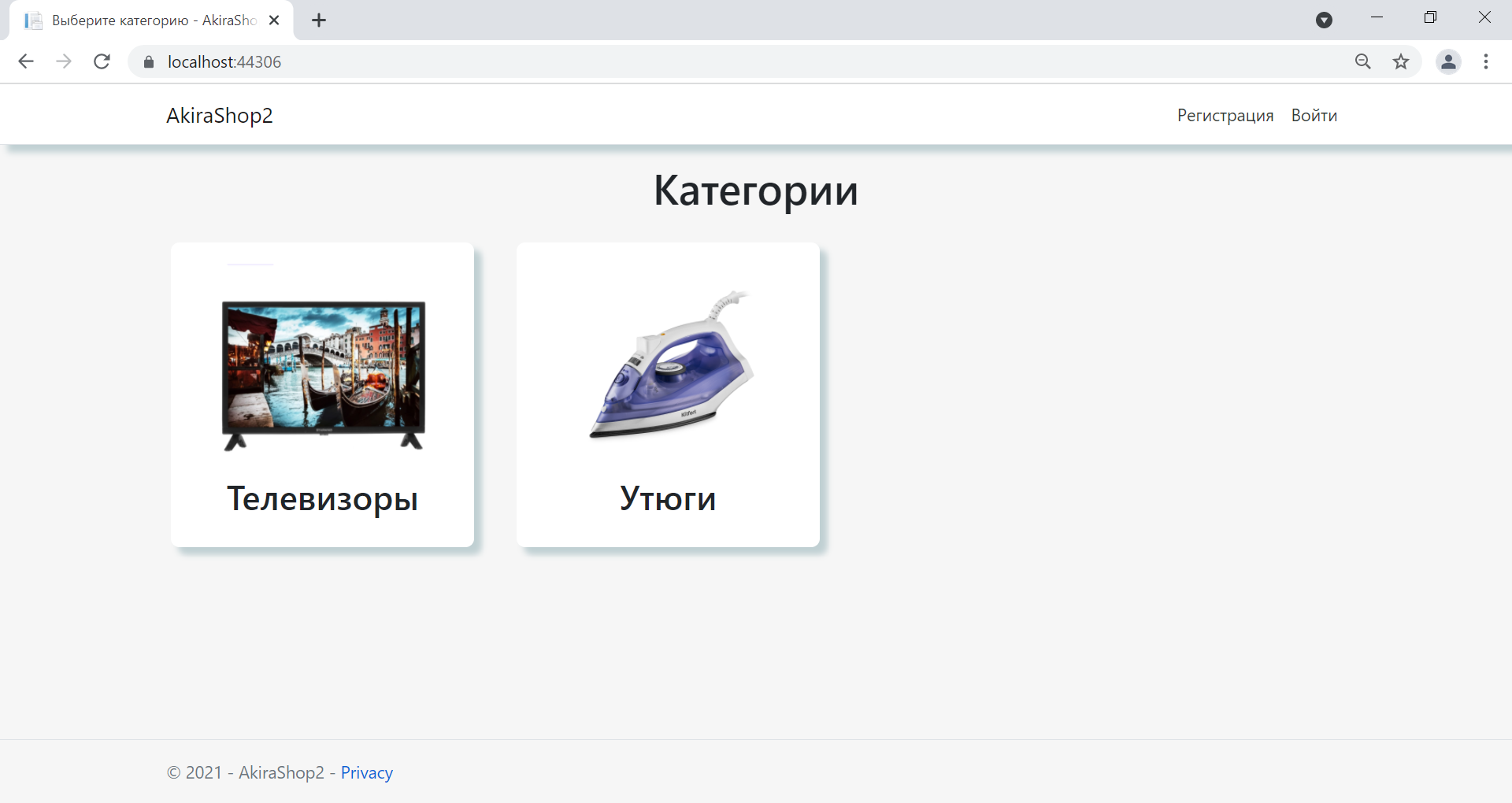


Рисунок 3.4.1 – Категории товаров

* Страница для просмотра товаров и фильтрации определенной категории (Рисунок 3.4.2).

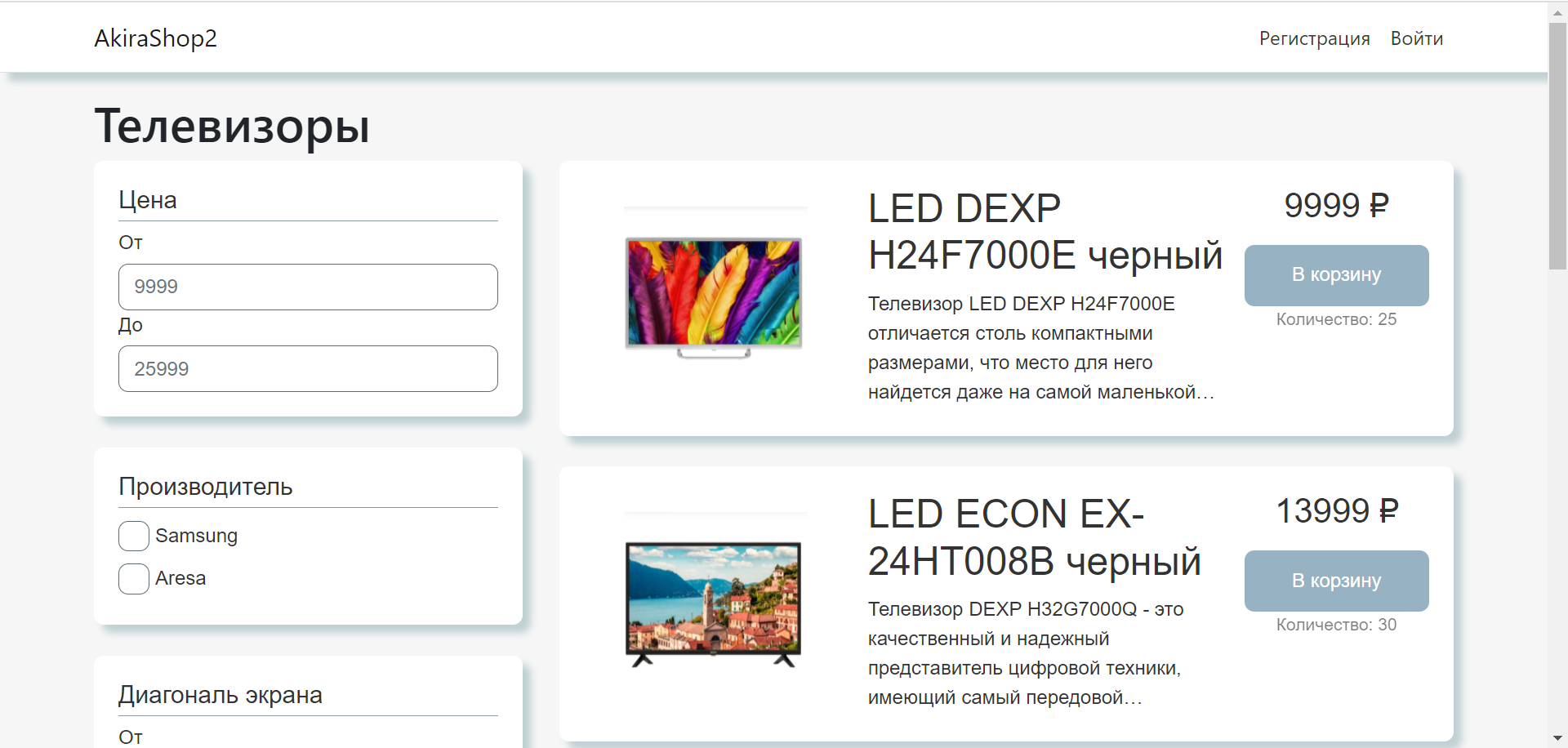


Рисунок 3.4.2 – Список товаров категории “Телевизоры”

* Страница для просмотра корзины и избранного (Рисунок 3.4.3). Аналогично выглядит страница готовых заказов. Доступно любому авторизованному пользователю.

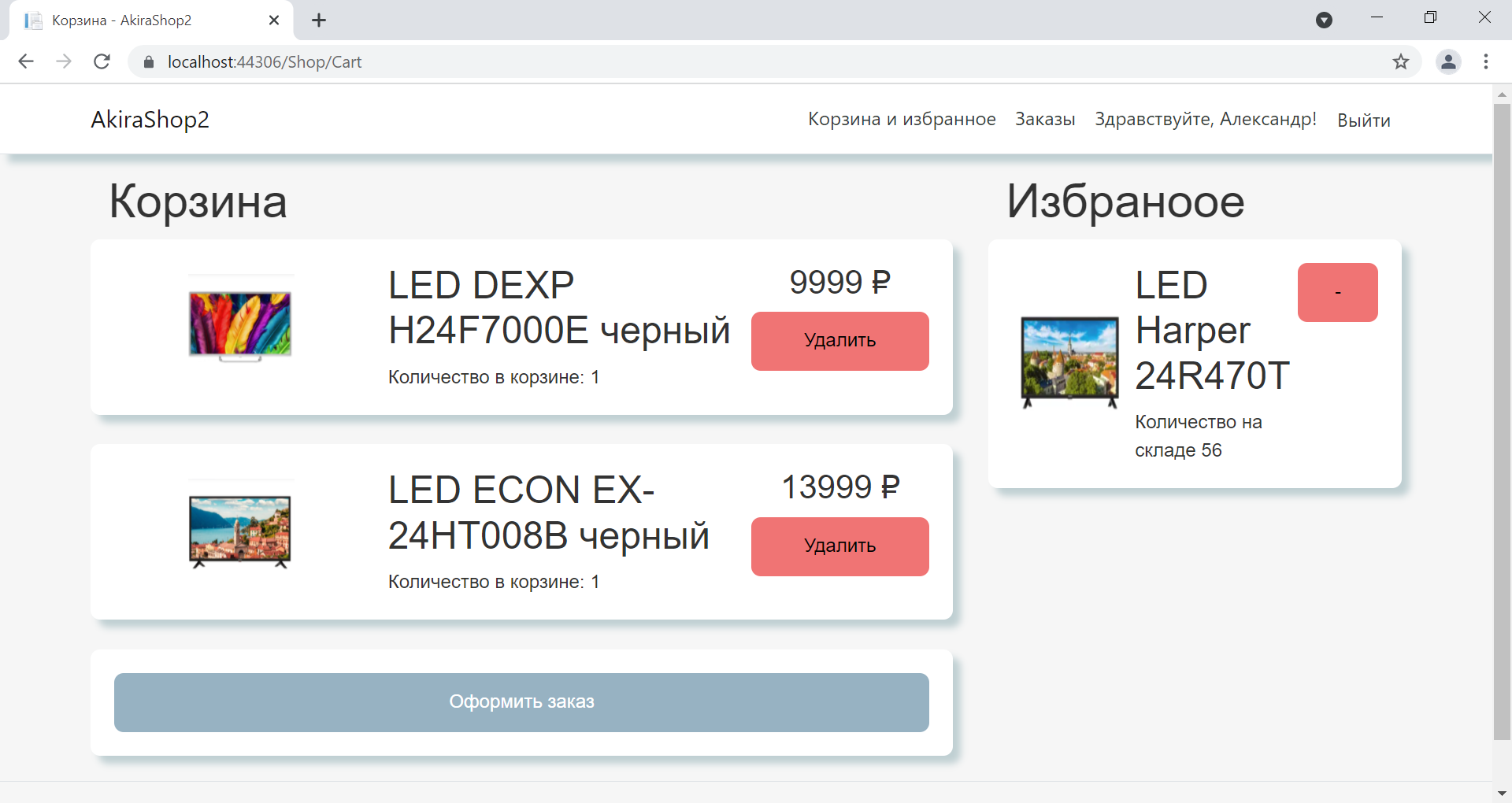


Рисунок 3.4.3 – Корзина и избранное

* Модальная страница формы авторизации пользователя (Рисунок 3.4.4) и регистрации (Рисунок 3.4.5). Данная страница появляется поверх основных страниц.

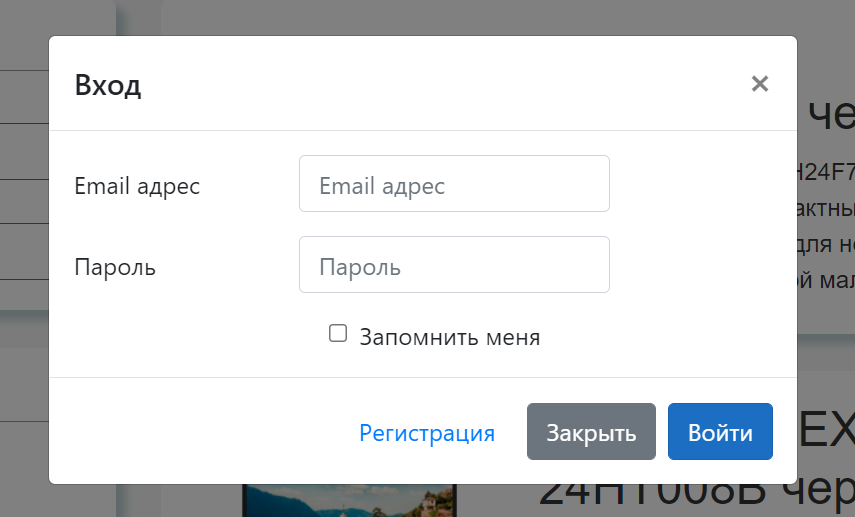


Рисунок 3.4.4 – Форма авторизации пользователя

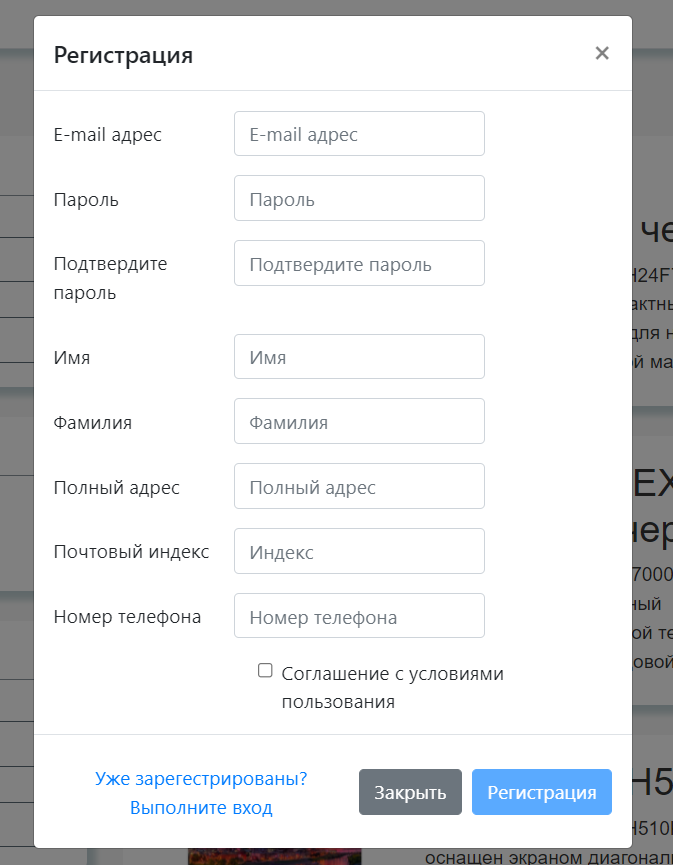


Рисунок 3.4.5 – Форма регистрации пользователя

* Страница добавления, изменения категории (Рисунок 3.4.6) и товаров (Рисунок 3.4.7). Страница производителей - аналогична предыдущим. Данные страницы доступны администратору.

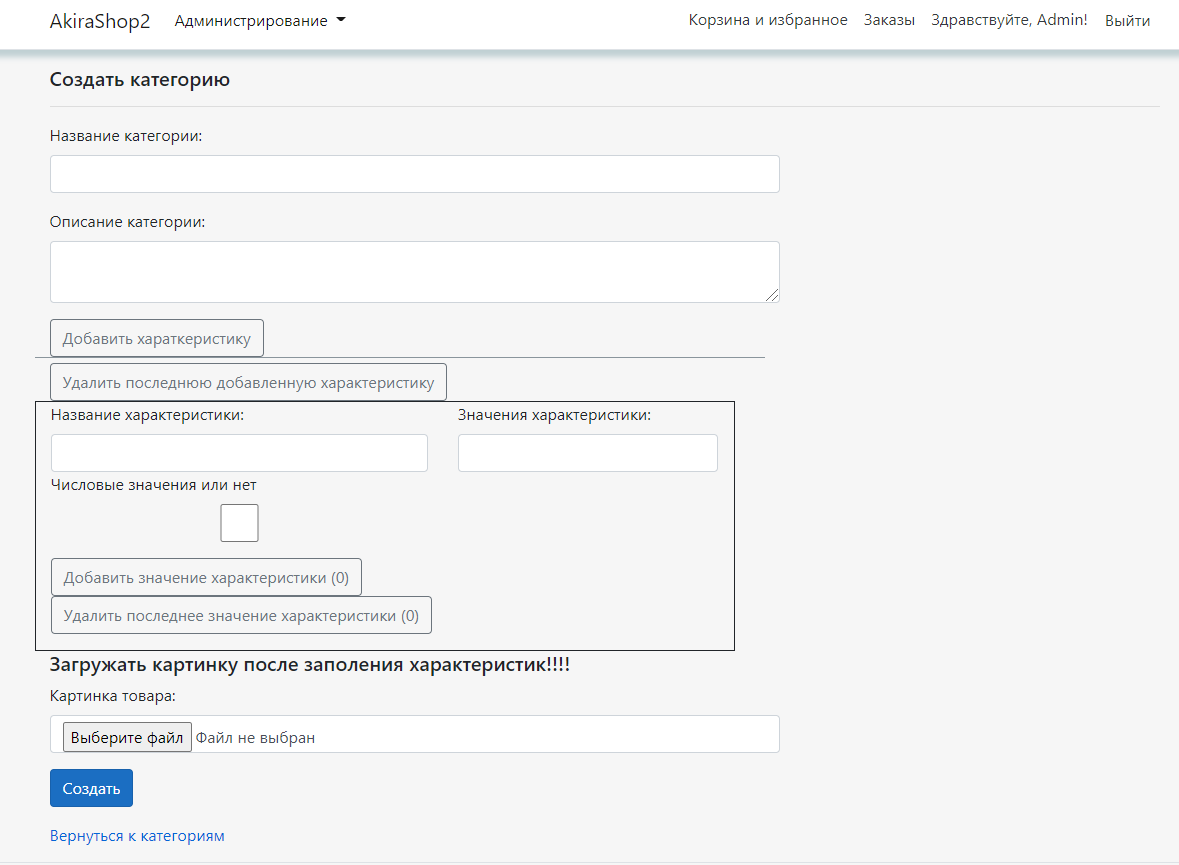


Рисунок 3.4.6 – Страница создания/изменения категории

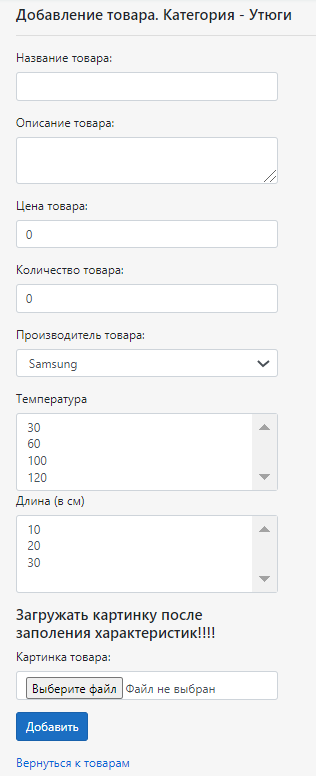


Рисунок 3.4.7 – Страница создания/изменения товара категории “Утюги”

* Страница просмотра товаров категории (Рисунок 3.4.8). Страницы просмотров категории, пользователей, производителей и заказов аналогичны. Данные страницы доступны администратору.

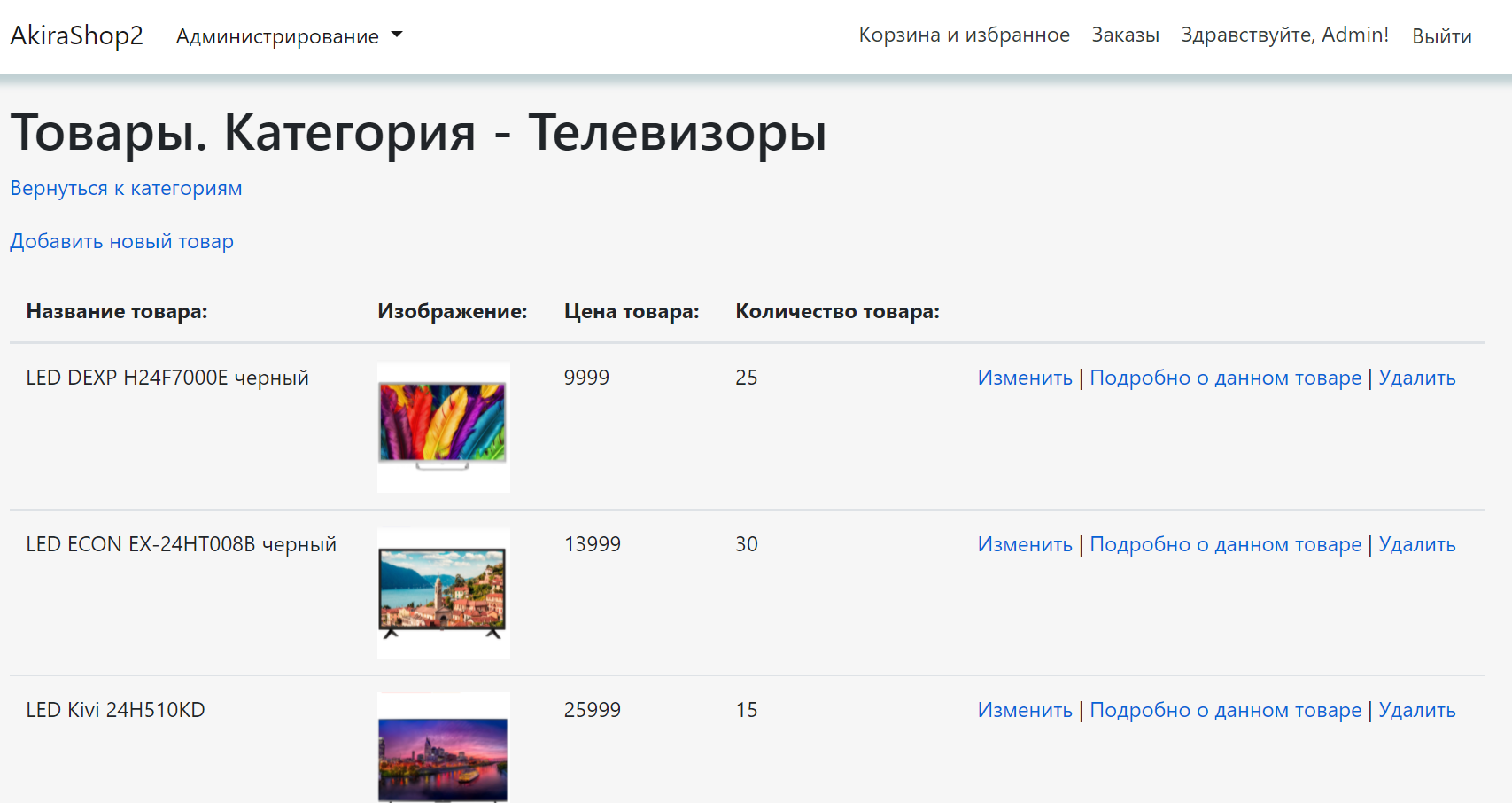


Рисунок 3.4.8 – Страница просмотра товаров категории “Телевизоры”

* Страница поиска по конкретным значениям (Рисунок 3.4.9). Доступна администратору.

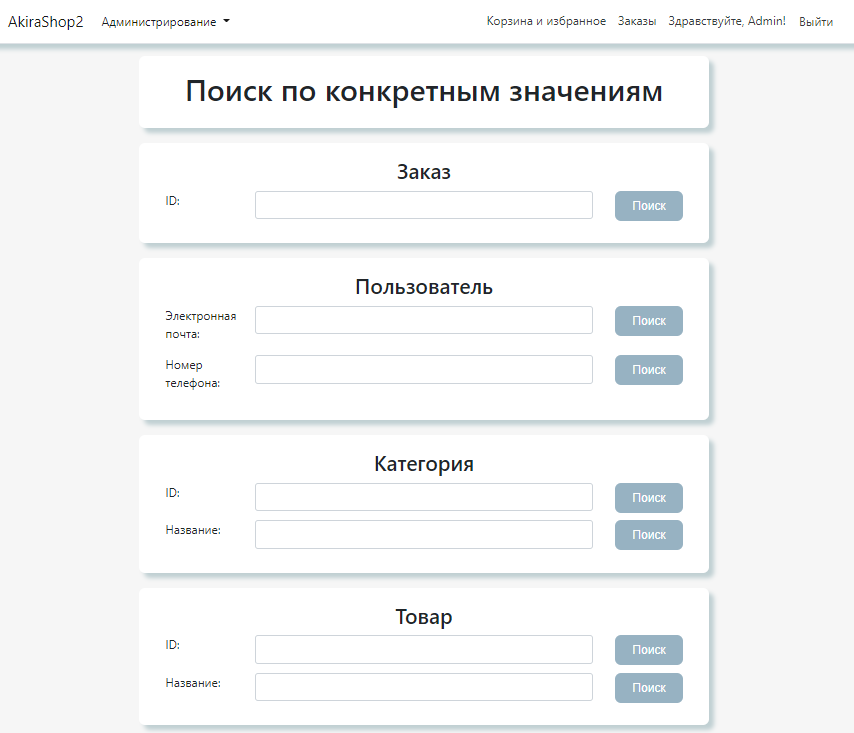


Рисунок 3.4.9 – Поиск по конкретным значениям

* Страница изменения пользовательской информации (Рисунок 3.4.10). Доступна любому авторизованному пользователю.

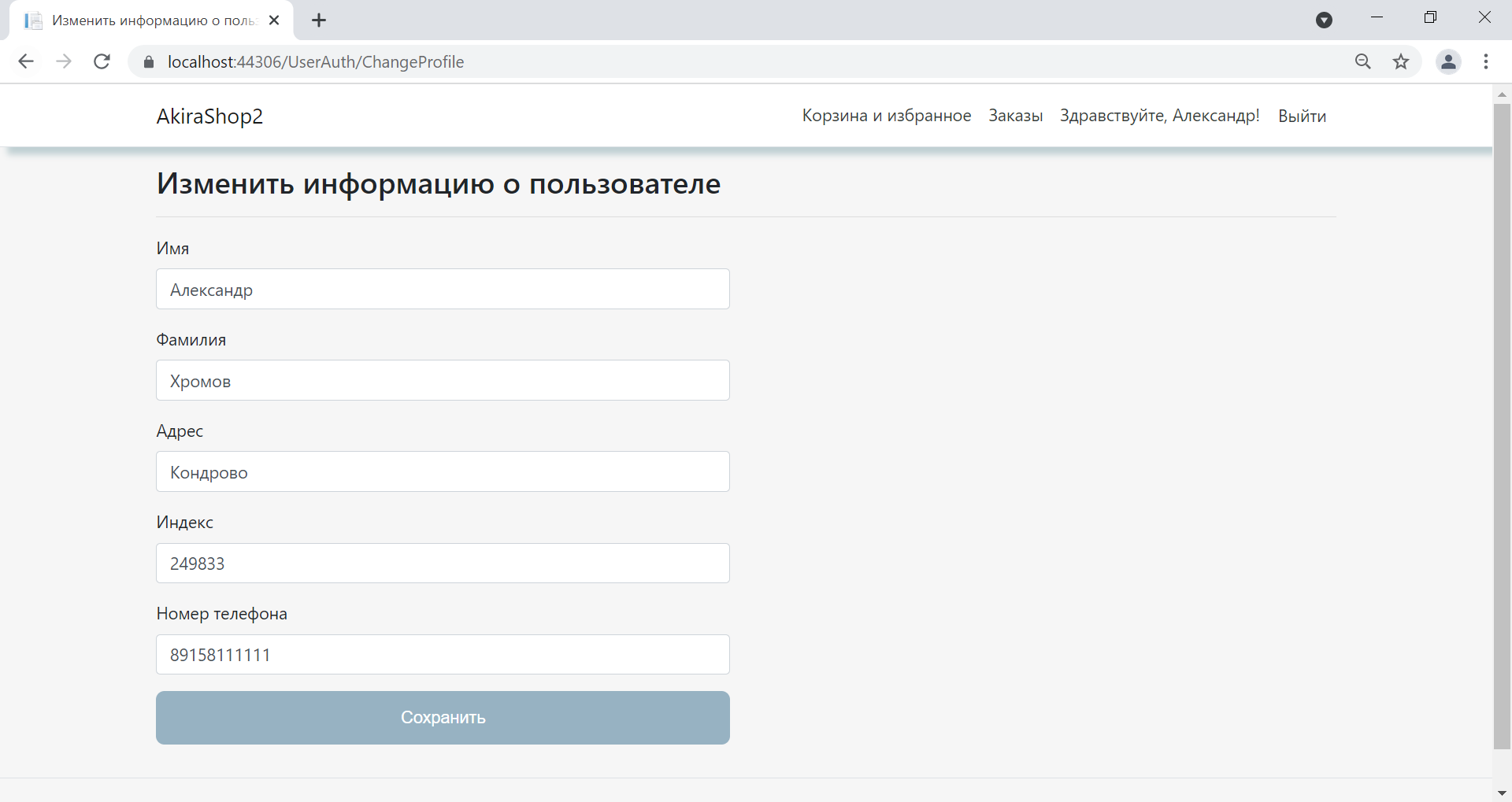


Рисунок 3.4.10 – Изменение пользовательской информации

В АРХИТЕКТУРУ

* 1. Разработка архитектуры приложения.

Было решено создавать именно веб-приложение, так как оно более универсально и практично для конечного пользователя. Веб-приложением можно пользоваться из любого браузера на любой операционной системы.

Для данного проекта подошла бы любая платформа разработки веб-приложения, была выбрана, как уже указывалось в исследовательской части, ASP.NET Core. Данная платформа выбрана из-за того что язык C# является уже хорошо знакомым. В следствии этого во время написания кода почти не возникло проблем.

Частью платформы .NET является Entity Framework. Данный ORM-фреймворк тоже имеет ряд преимуществ. Он позволяет создавать таблицы базы данных и саму базу данных напрямую из кода проекта. Entity Framework выполняет соответствующий запрос в базу данных, а затем предоставляет результаты в экземплярах объектов веб-приложения, для того чтобы далее с ним было проще работать. Данный фреймворк позволяет выполнять запросы LINQ к базе данных.

В данном проекте используется СУБД Microsoft SQL Server. Данная СУБД не особо отличается от уже изученной PostgreSQL. Так же данная СУБД уже является частью ASP .NET и не требуется устанавливать дополнительные библиотеки для другой СУБД.

Передача информации между компонентами внутри приложения осуществляется, как уже говорилось в исследовательской части, с использование MVC (Model-view-controller). Контроллеры используют REST архитектуру. В некоторых случаях (например при фильтрации) используется модель поведения PRG (Post-Redirect-Get), которая защищает пользователя от повторной отправки данных веб-форм, и значительно улучшает восприятие приложения пользователем.

* 1. Реализация функционирующего приложения

Описание сущностей базы данных:

‘Category’ – сущность для категорий. Имеет поля:

* Id – PRIMARY KEY, int, Не NULL
* Title – nvarchar(200), Не NULL
* Description – nvarchar(200), Не NULL
* Image – nvarchar(max), NULL
* Json\_Characteristics\_model – nvarchar(max), NULL

Последнее поле используется для фильтрации и имеет вид JSON: {“НазваниеХарактеристики” : { “Значения”: [“значение1”, “значение2”, “значение3”, …], “ДиапазонИлиНет”: true/false } }

Далее данное поле используется вместе с сущностью ‘Item’ для заполнения характеристик товара и фильтрации по ним.

‘Item’ – сущность для товара. Имеет поля:

* Id – PRIMARY KEY, int, Не NULL
* Title – nvarchar(200), Не NULL
* Description – nvarchar(1000), Не NULL
* Image – nvarchar(max), NULL
* Price – bigint, Не NULL
* Json\_Characteristics\_exact – nvarchar(max), NULL
* Amount – bigint, Не NULL
* CategoryId – FOREIGN KEY => Category.Id, int, Не NULL
* ManufacturerId – FOREIGN KEY => Manufacturer.Id, int, Не NULL

Поле Json\_Characteristics\_exact используется, как уже говорилось выше, для фильтрации товара. Имеет вид JSON: { “Характеристика1” : ”Значение1”, “Характеристика2” : ”Значение2”, “Характеристика3” : ”Значение3”, …}

Значения и характеристики соответствуют полю Json\_Characteristics\_model в сущности ‘Category’.

‘Manufacturer’ – сущность для производителя. Имеет поля:

* Id – PRIMARY KEY, int, Не NULL
* Title – nvarchar(200), Не NULL
* Description – nvarchar(200), Не NULL

‘Order’ – сущность для заказа. Имеет поля:

* Id – PRIMARY KEY, int, Не NULL
* UserOrderId – FOREIGN KEY => AspNetUsers.Id, navchar(450), Не NULL
* Status – nvarchar(max), Не NULL
* AdditionalInfo – nvarchar(max), NULL
* TotalPrice – bigint, Не NULL

‘OrderItem’ – дополнительная сущность для осуществления связи “многое-ко-многим” сущностей Order и Item и для хранения количества каждого товара в заказе. Имеет поля:

* Id – PRIMARY KEY, int, Не NULL
* OrderItem\_OrderId – FOREIGN KEY => Order.Id, int, Не NULL
* OrderItem\_ItemId – FOREIGN KEY=> Item.Id, int, Не NULL
* OrderItem\_Amount – int, Не NULL

‘AspNetUsers’ – сущность пользователя. В сущности содержатся стандартные поля ASP .NET фреймворка. К ним были добавлены дополнительные поля. Перечислим только значимые поля данной таблицы:

* Id – PRIMARY KEY, nvarchar(450), Не NULL
* Email – nvarchar(256), NULL
* PhoneNumber – nvarchar(max), NULL
* Address – nvarchar(250), NULL
* FirstName – nvarchar(250), NULL
* LastName – nvarchar(250), NULL
* PostCode – nvarchar(50), NULL

‘AspNetRoles’ – сущность ролей пользователей. В сущности содержатся стандартные поля ASP .NET фреймворка. Перечислим только значимые поля данной таблицы:

* Id – PRIMARY KEY, nvarchar(450), Не NULL
* Name – nvarchar(256), NULL

‘AspNetUserRoles’ – дополнительная сущность для осуществления связи “многое-ко-многим” сущностей AspNetUsers и AspNetRoles. Имеет поля:

* UserId – PRIMARY KEY, FOREIGN KEY => User.Id, nvarchar(450), Не NULL
* RoleId – PRIMARY KEY, FOREIGN KEY => Role.Id, nvarchar(450), Не NULL