Титульник

Оглавление

[Введение 3](#_Toc59377380)

[Технологический стек 5](#_Toc59377381)

[Сущности 9](#_Toc59377382)

[Архитектура 16](#_Toc59377383)

[Чат и SignalR 20](#_Toc59377384)

[Асинхронность 24](#_Toc59377385)

[Реализация системы 25](#_Toc59377386)

# Введение

В современном мире очень большую роль играет скорость решения тех или иных вопросов. Автоматизировать задачи – значит перенести ответственность за их частичную или полноценную реализацию на плечи электронно-вычислительных машин. Когда речь заходит о какой-либо продаже, очень важно уметь вовремя решать вопросы и по возможности автоматизировать рутинные процессы.

Целью нашей работы будет написание программы, которая будет помогать в продаже косметических товаров.

Исходя из цели нашей работы, определим ряд задач, которые она должна решать:

В первую очередь необходимо ознакомиться с техническими инструментами, с помощью которых мы можем реализовать программу в жизнь в целом, без привязки к отдельному функционалу. Следует понять, какой подход нам подойдет лучше.

Во-вторых, следует проработать структуру хранения данных. Определить ключевые понятия и выделать некие сущности.

В-третьих, мы должны построить гибкую и качественную программную архитектуру, которая бы могла отзывчиво отвечать на возможные изменения в будущем, которые так или иначе подразумеваются в связи с очень живой и подвижной спецификой предметной области программного обеспечение данной работы.

В-четвертых, графический пользовательский интерфейс должен быть доступен к пониманию рядовому пользователю, быть достаточно простым и обладать приятным внешним видом.

Пятая задачу, которую следует решить, это коммуникация между администратором приложения и клиентами. Другими словами это можно назвать некоторой поддержкой покупателей, когда им необходимо уточнить или узнать информацию, скажем, о наличии какого-либо товара.

# Технологический стек

Основой любого программного обеспечения являются технологии, на которых оно разрабатывается и проектируется. Достаточно много факторов зависит от того, что выберет команда разработки и проектирования на начальных этапах.

Одни технологии хороши для научно-исследовательской работы, другие хороши для прикладных повседневных задачах. При большом желании можно использовать первое для целей второго, но может получится что мы будем, как говорится, забивать гвозди микроскопом. Поэтому следует выбирать тот стек технологий, который чаще всего решает именно такие задачи. Под технологическим стеком подразумевают перечень технологических решений, из которого складывается программное обеспечение.

Несмотря на то, что технологии должны часто использоваться для решения сходных задач, они также должны иметь хорошую перспективу для развития. Это говорит о том, что наши системы будут идти в ногу со временем. В худшем случае это приведет к частичной или полной смене стека, а это влечет к временному застою в развитии систему потому, что придется искать новых специалистов, либо переучивать текущих.

Отсюда следует следующий фактор : технологии должны иметь широкое комьюнити разработчиков и технических специалистов вокруг себя. Это нужно для того, чтобы быстро находить ответы по вопросам использования технологии во время разработки и возникновения конфликтов во время эксплуатации. Одно из золотых правил разработки гласит, что не нужно изобретать велосипед, ведь кто-то уже сталкивался с этой проблемой и нашел ее лучшее решение. Это чаще всего работает. Чем больший круг людей активно пользуется, например, выбранным языком программирования, тем больше людей сталкивалось и решало эту проблему.

Определим также целевую платформу разработки. Программное обеспечение зависимо непосредственно от того, под какое аппаратное обеспечение оно создается. Поскольку нам необходим функционал электронного магазина косметических товаров, то формат настольного приложения, например, под Windows не походит по ряду причин. Во-первых, покупатели вынуждены пользоваться персональным компьютером и устанавливать программное обеспечение себе на машину. А что если на пользуются персональным компьютером на базе Mac? Придется писать дополнительное программное обеспечение под эту операционную систему, а потом и под другие. Потенциальный покупатель может пользоваться чем угодно. Во-вторых, если кому-то нужно посмотреть один товар, то ему будет неудобно тратить время на установку приложения, ему нужен быстрый доступ. Это второй фактор против приложения на базе desktop решений.

Если нам нужен доступ к системе в любой момент времени, то хорошо подойдет решение на базе мобильных приложений. Современный человек очень много времени проводит, держа мобильное устройство в руках, и оно почти всегда доступно для использования. Конечно для использования такого программного обеспечения его тоже нужно устанавливать, но в целом приложения более легковесны и их установка проще и прозрачнее для конечного пользователя. Количество целевых платформ сокращается по большому счету до двух – iOS и Android. Проблема разработки под разные системы конечно остается, но на рынке доступны технологии, которые позволяют писать один код под две разные системы одновременно. Правда есть один нюанс, связанный с iOS : если мы разрабатываем на Windows, то сборка приложения под яблочное устройство будет невозможно без Mac-устройства.

В принципе разработка под мобильное приложение идея неплохая, но все же следует его рассматривать, как дополнение к основной информационной системе. Решение, которое позволяет отвязаться от конкретной платформы – это web-приложение. Конечные пользователи должны будут лишь иметь предустановленную программу для общения данные через сеть – web-браузер. Web-браузер установлен на большинстве платформ, что позволяет пользователю свободно пользоваться системой независимо от устройства, будь то персональный компьютер или смартфон, или даже телевизор или игровая консоль.

Web-разработка состоит из двух частей – frontend и backend разработок. Зона ответственности frontend-разработки – клиентская часть, то есть непосредственно то, с чем взаимодействует пользователь. Backend-разработка – это непосредственная работа с «сырыми» данными, их обработка и построения взаимодействия. Это та «подкапотная» работа, которую не видит пользователь. Работа с базой данных и формирование на основе них моделей, которые описывают предметную область программного обеспечения. Backend предоставляет интерфейс взаимодействия с данными.

Основной платформой нашего приложения будет являться .NET Core 3.1. Это технология от компании Microsoft, которая позволяет разрабатывать приложения, которые можно запускать на разных системных платформах. То есть при желании мы можем развернуть основу нашего приложения как на Windows, так и на Linux. .Net Core является передовой технологией, которая развивается и будет актуальна ближайшие годы. Она предоставляет широкий перечень решений, который будет нам полезен. Первая – это непосредственно основа Web-приложения – Asp.Net Core MVC. Это позволяет нам писать как backend, так и frontend, используя razor-pages в качестве основы для клиентской части. Не менее полезным для нас будет ORM технология EntityFrameworkCore, речь о которой будет идти ниже. Эта технология позволяет писать взаимодействие с базой данных непосредственно на выбранном языке программирования и не писать сами запросы на языке SQL. Язык программирования, который мы будем использовать – C#. Язык является передовым во многих аспектах, его синтаксис насыщен так называемым «сахаром», который делает код прозрачнее, но инкапсулирует либо сложные конструкции, либо более низкоуровневые операции. Сервером базы данных будет служить SQL Server, СУБД – MSSQL. На клиентской части используется HTML, CSS и JavaScript. Для упрощения работы со стилями используется библиотека Bootstrap 4, а для простого взаимодействия между деревом DOM и языком программирования JavaScript используется библиотека JQuery.

Для коммуникации, о которой шла речь в введении, используется библиотека SignalR, а для авторизации и аутентификации – Identity.

# Сущности

В первую очередь систему определяют сущности, которые отражают предметную область системы. Мы используем C# в связке с ORM EntityFrameworkCore, поэтому опишем сущности в виде классов на соответствующем языке программирования.

Первая сущность, на которой следует остановить внимание – это сущность пользователя приложения. Обратим внимание, что здесь не хранится информация о пароле так, как за авторизацию и аутентификацию отвечает библиотека Identity, которая разворачивает под себя отдельную базу данных, где скрывает за туманном хеша данные о пароле.

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| Id | Guid | Уникальный идентификатор сущности |
| Email | String | Отражает информацию о почтовом адресе пользователя |
| Phone | String | Отражает информацию о номере телефона пользователя |
| FullName | String | Отражает информацию о полном имени пользователя |

Сделаем замечание, что Email является логином внутри нашей системы.

Поскольку мы имеем дело с товарами, то они должны обладать некоторыми категориями. Опишем сущность категории.

|  |  |  |
| --- | --- | --- |
| Id | Guid | Уникальный идентификатор сущности |
| Title | String | Отражает информацию о названии категории |
| ParentCategoryId | Guid | Содержит внешний ключ на родительскую категорию |

Обратим внимание, что категории могут содержать вложенность один в другую, поэтому храним ключ на родительскую категорию.

Следующая сущность ключевая – это сущность продукта.

|  |  |  |
| --- | --- | --- |
| Id | Guid | Уникальный идентификатор сущности |
| CategoryId | Guid | Содержит внешний ключ на категорию, которой принадлежит товар |
| Description | String | Отражает описание товара |
| Price | Description | Отражает информацию о стоимости товара |
| Title | String | Отражает информацию о названии товара |

С понятием товара тесно связаны фотографии, которые его визуализируют. Поэтому опишем сразу две сущности : первая является непосредственно сущность фотографии, а вторая связующее звено между продуктом и его фотографиями.

|  |  |  |
| --- | --- | --- |
| Id | Guid | Уникальный идентификатор сущности |
| Photo | String | Отражает информацию о base64 представлении фотографии |

Таблица связи фотографий и продукта.

|  |  |  |
| --- | --- | --- |
| Id | Guid | Уникальный идентификатор сущности |
| PhotoId | Guid | Отражает информацию о внешнем ключе на фото |
| ProductId | Guid | Отражает информацию о внешнем ключе на продукт |
| DisplayOrder | Int | Отражает информацию о порядковом номере фотографии при отображении |

Следует остановиться на том моменте, как мы храним фотографии. Фотографии хранятся непосредственно в базе данных в формате base64. Это позволяет хранить данные централизовано и вместе.

Нам необходимо хранить информацию о том, в каком количестве храниться тот или иной товар. Введем сущность хранилища.

|  |  |  |
| --- | --- | --- |
| Id | Guid | Уникальный идентификатор сущности |
| ProductId | Guid | Отражает информацию о внешнем ключе на продукт |
| Count | Int | Отражает информацию о количестве товара |

Для обработки заказа пользователем введем сущность корзины пользователя.

|  |  |  |
| --- | --- | --- |
| Id | Guid | Уникальный идентификатор сущности |
| ProductId | Guid | Отражает информацию о внешнем ключе на продукт товара в корзине |
| UserId | Guid | Отражает информацию о внешнем ключе на пользователя товара в корзине |
| Status | Enum.BasketStatus | Отражает информацию о статусе товара в корзине |

Статус товара в корзине описывает с помощью перечисления BasketStatus, которая имеет два состояния : Hold и Paid. Hold – значит, что товар был добавлен в корзину, но не оплачен, Paid – значит, что товар был оплачен и был произведен вычет по товару из хранилища.

Чтобы в нашей системе был доступен обмен сообщениями, мы используем библиотеку SignalR. Но данная библиотека позволяет лишь реализовать механизм обмена сообщениями, но не хранит информацию о них. Поэтому в рамках обработки сообщения мы должны сохранять данные в базу данных. Введем сущность, которая отвечает за сообщения.

|  |  |  |
| --- | --- | --- |
| Id | Guid | Уникальный идентификатор сущности |
| From | String | Отражает информацию об отправителе сообщения (логин) |
| To | String | Отражает информацию о получателе сообщения (логин) |
| Date | DateTime | Отражает информацию о времени отправки сообщения |
| MessageBody | String | Отражает тело сообщения |

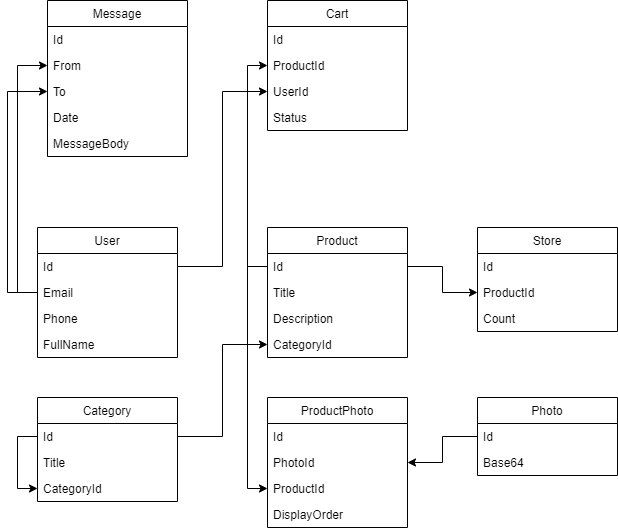


Рисунок Схема зависимостей сущностей

# Архитектура

Перед непосредственным написанием исходного кода необходимо проработать такое понятие, как архитектуру приложения. Архитектура приложения – это то, каким образом будут скомбинированы и как буду взаимодействовать его элементы на разном уровне.

В задачах к работе мы обозначили, что делаем на ставку на гибкость и изменчивость систему. Поэтому обратимся к лучшим практикам разработки информационных систем. Они гласят, что программировать следует на уровне абстракций, а не на уровне реализации. Это значит, что обозначая какое-либо взаимодействие между отдельными элементами в коде, мы должны опираться на схему взаимодействия, либо, другими словами, на то, как «общаются» между собой эти элементы. Если мы говорим про какой-то метод, что мы должны четко понимать, что он возвращает, что он принимает, какой тип данных он представляет. Таким образом мы становимся независимы от реализации. Мы знаем, что есть необходимый для нам механизм, и нам неважно, как он работает внутри, нам нужно лишь то, что он производит. Такой подход называется слабосвязанный.

Далее нам необходимо выделить три глобальных слоя приложения : слой непосредственной работы с данными, слой обработки этих данных в том ключе, который подразумевают требования и самый высокий слой – клиентский.

Первый слой, назовем его Data Access Layer, отвечает за подключение к базе данных, получению и изменению данных в ней. Так как мы решили писать код на абстрактном уровне, то все взаимодействие мы опишем шаблоном проектирования Репозиторий на основе Generic интерфейсов. Это значит, что мы напишем универсальные методы CRUD (Create-Read-Update-Delete) операций, которые не будут зависеть от конкретной сущности или таблицы, а будут уметь работать со всеми. Благодаря такому подходу мы не будем обязаны каждый раз при добавлении новой сущности сызнова описывать эти тривиальный операции. Но в тоже время, чтобы оптимизировать некоторые особенные запросы или модернизировать чтение на основе иных полей, мы можем наследовать интерфейс от интерфейса базового репозитория и расширить его. Таким образом мы следуем еще одному принципу разработки «Код должен быть открыт для расширения и закрыт для изменения». За реализацию интерфейса будет отвечать библиотека ORM технологии EntityFrameworkCore. Удобство такого подхода заключается еще и в том, что если с течением времени нам будет необходимо использовать другую ORM, то мы просто меняем ее и переписываем реализацию репозитория, но все взаимосвязи внутри всего приложения остаются неизменными, они об этом не знают. Они знают лишь то, что есть какое-то хранилище данных, с которым можно взаимодействовать. Причем этим хранилищем необязательно должна быть реалиционная база данных. Мы можем реализовать репозиторий так, как захотим. Будь то хранение данных в файловой системой на основе XML, либо просто один большой txt файл, с которого мы читаем все данный.

На данном этапе мы вынесли в отдельные проекты библиотеку классов Data Access Layer, библиотеку интерфейсов, а также библиотеку сущностей, которая по сути отражает таблички в базе данных и была описана в главе «Сущности».

Следующий этап – описание так называемой бизнес логики в слое Business Logic Layer. Этот слой комбинирует сущности из слоя доступа к данным в модели, которые уже используются дальше в клиентском слое. В этом слое хранится код, который составляет весь движок приложения. Здесь мы можем прописывать сложную валидацию, либо настраивать триггеры на определенные события, запускать код по таймеру и прочее и прочее. Поэтому мы создаем два дополнительных проекта библиотеки классов : BLL и модели BLL.

Отдельный проект выделен под работу с библиотеку Identity. В ней хранится контекст подключения к базе данных и модели расширения классов пользователя.

Следующий слоя – это последний слой, слой клиентский. Этот слой непосредственно является AspNetCore MVC приложением. Здесь реализован подход к разделению данных Model View Controller.

Слой View – это то, что непосредственно видит и с чем может взаимодействовать пользователь. Пользователь видит те Model, которые присылает ему Controller. Пользователь может изменять состояния в Model и отправлять их в Controller, далее может производится валидация, обращение к сервисам DLL и так далее.

Слой Model – это схематичное представление данных, которое будет удобно доставлять конечному пользователю. Эти модели не являются моделями бизнес логики и не являются моделями уровня доступа к данным – сущностям. В некоторых случаях они могут представлять одну и ту же модель данных и встанет вопрос, почему тогда не использовать одну модель вместо трех, не избыточно ли это. Но прелесть заключается в том, что если придется расширять любой из трех классов, то это не отразится на других двух, а если бы был один класс на все, это вызывало бы трудности.

Слой Controller – это точки входа на сервер, пользователь при переходе на страницу или отправки формы пользуется этими точками. По сути контроллер является первым постом для данных, тут может быть какая-то клиентская серверная валидация или что-то еще, но не слишком сложное. По сути методы контроллера являются диспетчерами интерфейсов бизнес логики.



Рисунок Схематичное представление MVC

В конечном итоге мы имеем 7 проектов, из которых состоит наше приложение : веб-приложение, интерфейсы, модели и BusinessLogicLayer, сущности и DataAccessLayer, Identity.

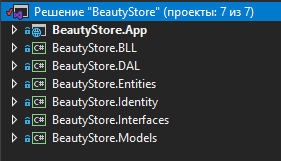


Рисунок Структура приложения

# Чат и SignalR

Одна из особенности функциональности, которая была оговорена в задачах к проекту требует реализовать механизм общения пользователей приложения с администратором приложения. Разделим этот механизм на несколько задач.

Первая задача – хранение истории сообщений. Это реализуется с помощью той архитектуры и технологий, которые были оговорены выше. Сообщение, поступив на бизнес-логику, снабжает сущности слоя представления всеми необходимыми данными и посредством реализации репозитория силами EntityFrameworkCore кладет в базу данных.

Вторая задача – отображение на клиентской части в удобном и красивом формате. Все это решается с помощью компоновки элементов и блоков на HTML и стилизацией CSS.

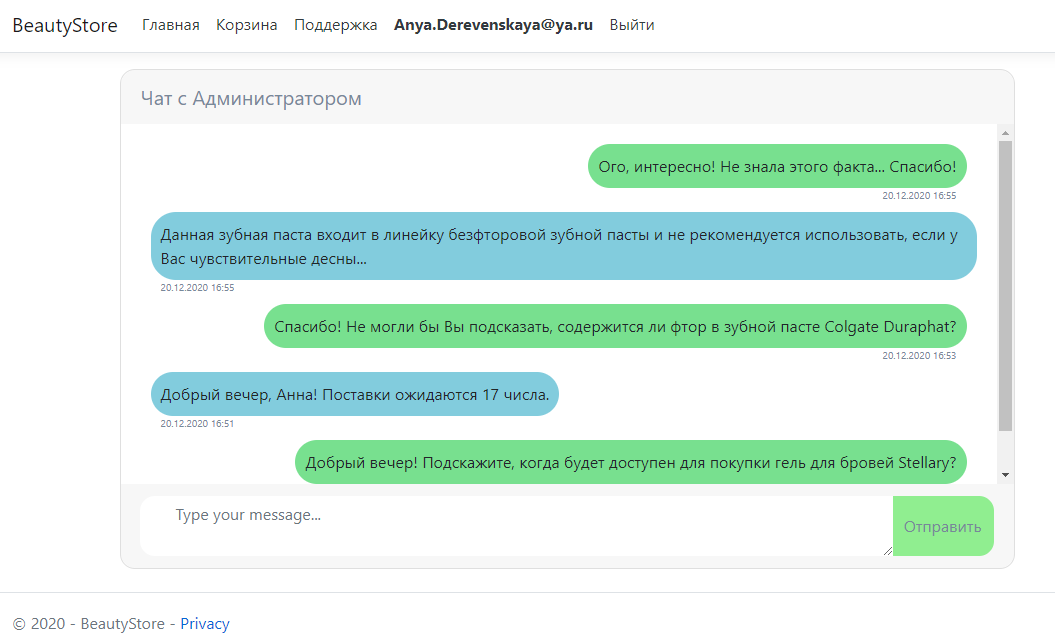


Рисунок Чат со стороны клиента

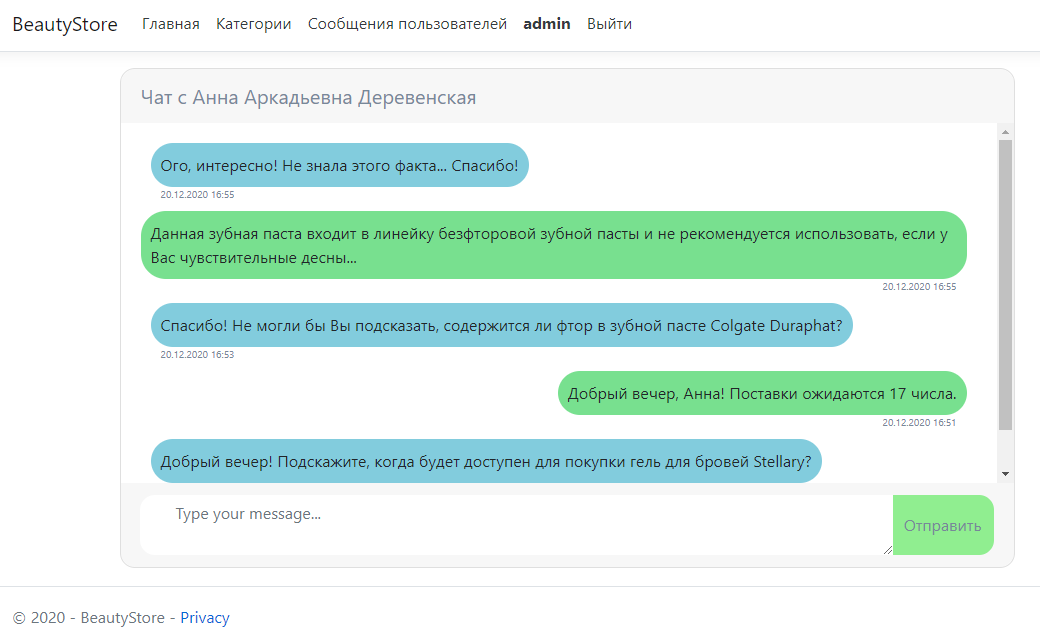
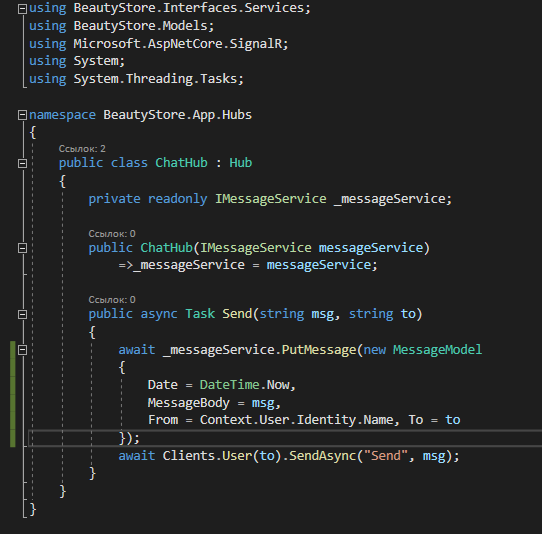


Рисунок Чат со стороны администратора

Третья задача – это организовать общения в режиме реального времени. Для этого используем библиотеку SingalR. Опишем, как он работает. SignalR может использовать в качестве транспорта и websockets, и longpolling. Транспорт можно задать, а можно оставить на откуп SignalR, который сам выберет нужный. В случае, если можно использовать websocket, то он будет работать через websocket, если такой возможности нет, то он будет спускаться дальше, пока не найдёт приемлемый транспорт.

Для работы SignalR опишем приемник на стороне сервера и код взаимодействия на стороне клиента.



Picture Код хаба на серверной стороне

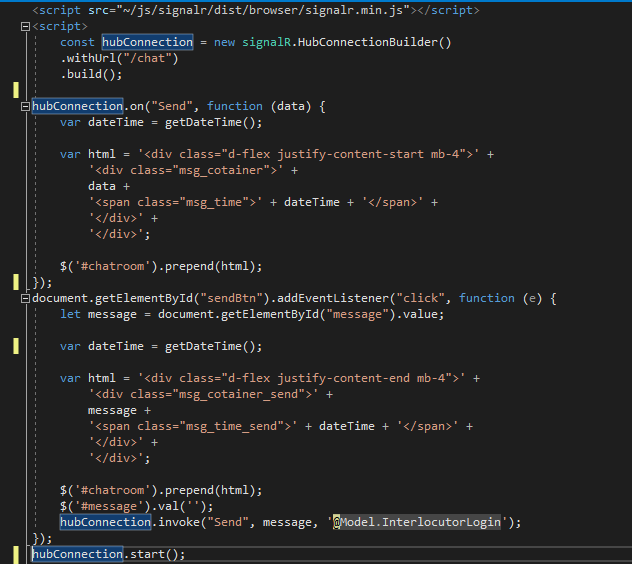


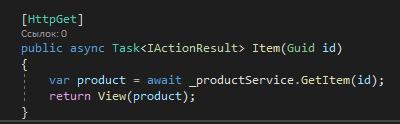
Рисунок Хаб на клиентской стороне – JavaScript

Получив уведомление на серверной стороне, мы записываем сообщение, адресата и получателя в базу. При отправке и получении сообщений на клиентской части, мы добавляем элементы на HTML в DOM, чтобы пользователь видел активность.

# Асинхронность

Язык программирования позволяет писать асинхронный код, который работает через использование ключевых слов async/await. Такой подход позволяет работать в стиле не блокирующем вызывающий поток. То есть с помощью асинхронности мы помечаем операции дожидаться окончания которых мы не собираемся. Это очень важно потому, что мы работаем с базой данных, ожидание действий которой может потребовать времени.

Используем асинхронность на всех уровнях.



Асинхронный метод контроллера

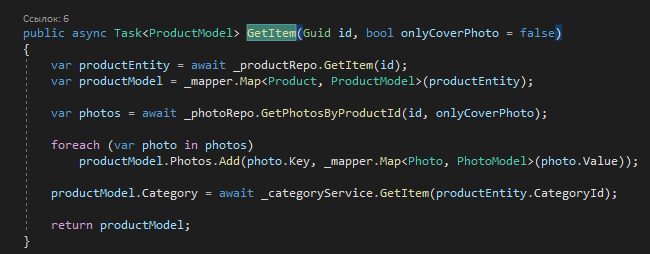


Рисунок Асинхронный метод BLL

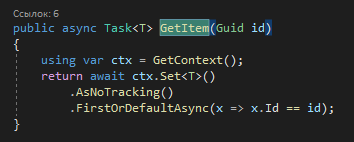


Рисунок Асинхронный метод DAL

# Реализация системы

В системе доступно три роли : неавторизованного пользователя, покупателя и администратора системы. Просмотр основной страницы доступен любому пользователю. Первое различие – шапка. Пользователь может посмотреть корзину и перейти в чат с поддержкой, администратор может перейти на страницу диалога со всеми пользователями и страницу редактирования категорий.

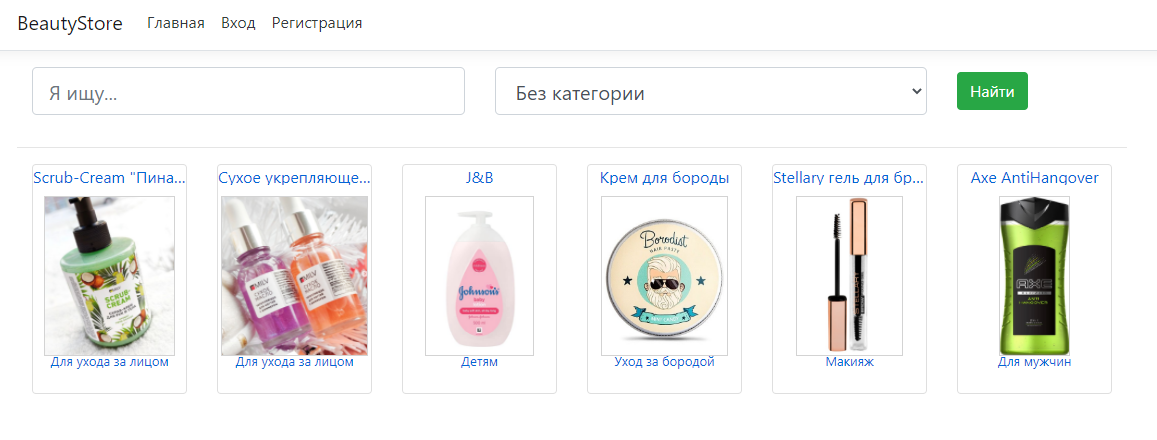


Рисунок Главная страница (пользователь не авторизован в системе)



Рисунок Страница товара от лица администратора

Если перейти на страницу товара, то администратору будет доступно редактирование товара и внесение факта поставки товара в систему. Также если количество фотографий товара превышает единицу, то будет доступна их прокрутка посредством стрелочек по бокам.

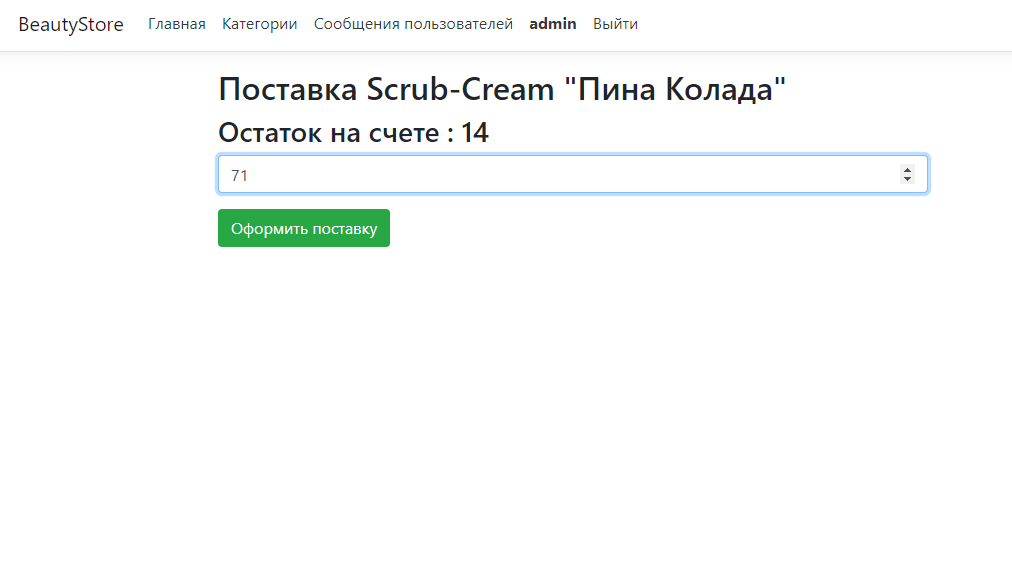


Рисунок Страница внесения информации о поставке

На странице поставки администратор может узнать количество товаров на складе и внести информацию о новой поставке.

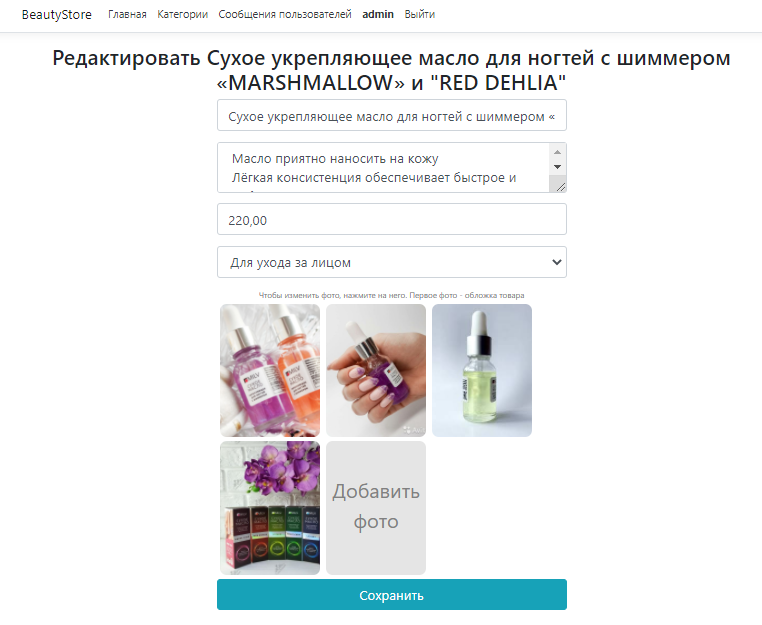


Рисунок Страница редактирования товара

На странице редактирования товара администратор может вносить изменения в данные о названии, описании, цене и категории товара. Также доступна множественная загрузка фотографий на сервер.

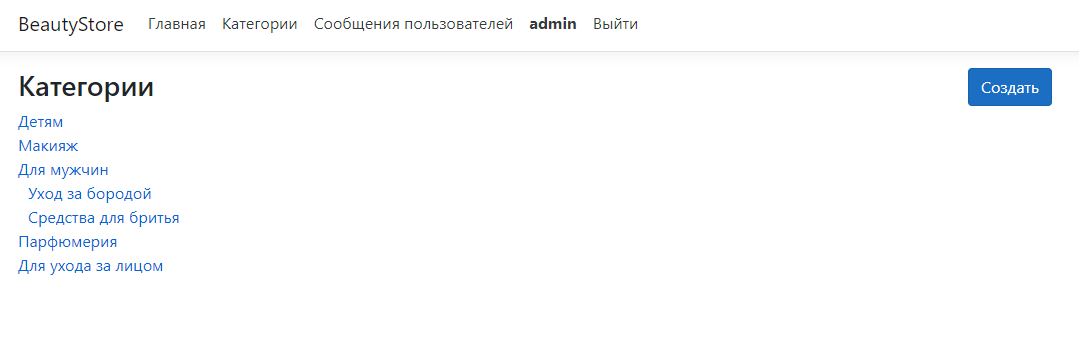


Рисунок Страница категорий

Администратор может видеть список категорий.

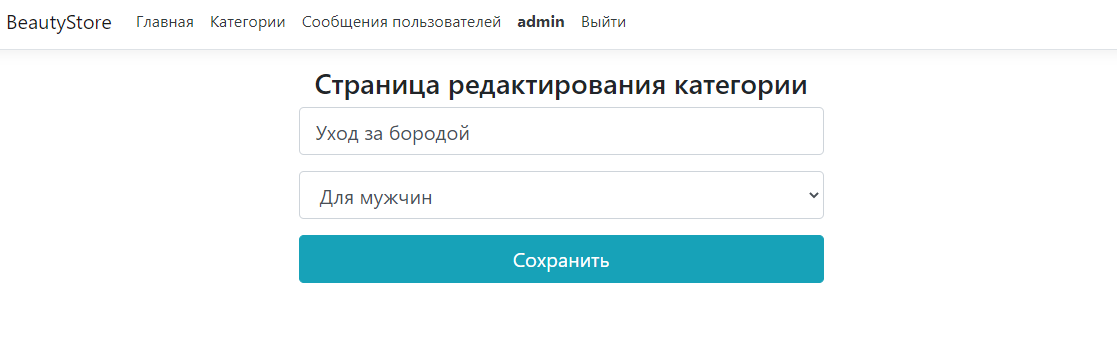


Рисунок Страница редактирования категории

На странице редактирования категории администратор может изменить родительскую категорию или название категории.

Если покупатель заходит на страницу товара, то он видит кнопку «В корзину». При нажатии на нее, либо по переходу пункта из пункта меню, пользователь попадает на страницу товаров в корзине.

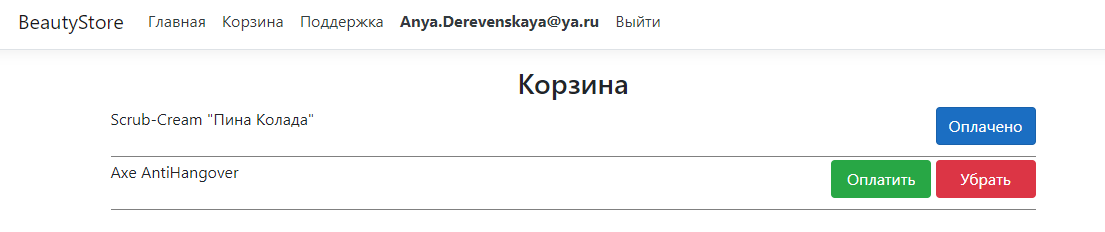


Рисунок Страница корзины пользователя

# Заключение

В результате работы была разработана информационная система, которая является хорошей основой для дальнейшего продолжения наращивания функционала. Были изучены методы гибкой разработки и разработки веб-приложений. Были использованы современные технологии, знания в которых будут полезны при дальнейшей работе над проектом.

# Список литературы

1. Обзор Web-разработки (<https://sky-rzn.ru/blog/interesnoe/chto-takoe-web/>)
2. Создание архитектуры программы (<https://habr.com/ru/post/276593/?_ga=2.126598857.533583176.1589107565-932482008.1578951813>)
3. Схема разделения данных Model-View-Controller (<https://javarush.ru/quests/lectures/questcollections.level06.lecture01>)
4. Реляционная база данных (<https://aws.amazon.com/ru/relational-database/>)
5. Руководство по Entity Framework Core (<https://metanit.com/sharp/entityframeworkcore/>)
6. Введение в ASP.NET Core Identity (<https://metanit.com/sharp/aspnet5/16.1.php>)
7. ASP.NET Core MVC (<https://metanit.com/sharp/aspnet5/3.1.php>)
8. SOLID (<https://medium.com/webbdev/solid-4ffc018077da>)
9. Dependency Injection (https://metanit.com/sharp/aspnet5/6.1.php)