

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia teaduskond

Informaatikainstituut

Tarkvaratehnika õppetool

Infosüsteem restoranidele, kasutades Microsoft
.NET tehnoloogiad

Bakalaureusetöö

Üliõpilane: Aleksander Semjonov

Üliõpilaskood: 093376IAPB

Juhendaja: Kaarel Allik

Lektor, tehnikateaduste magister

Tallinn

2015

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

.....

(kuupäev)

.....

(allkiri)

Infosüsteem restoranidele, kasutades Microsoft .NET tehnoloogiad

Annotatsioon

Peamine eesmärk on analüüsida probleemi ja rakendamist söögikohta infosüsteemi

Peamiseks eesmärgiks on analüüsida probleemi ja rakendamist söögikohta infosüsteemi

Lõputöö koosneb kahest põhiosast. Esimeses osas vaadeldakse ülesande püstitamist, andmete loogilise struktuuri ja tüüplahenduse võimalust. Teises osas vaadeldakse Windows Presentation Foundation ja ASP.NET MVC tehnoloogiat.

Selle töö peamiseks tulemuseks on prototüüpi rakenduse realiseerimine söögikohtades, kasutades WPF ja lisamine funktsionaalsuse (halduse) osa eraldi veebirakendusse.

Bakalaureusetöö on kirjutatud vene keeles, rakendus on inglise keeles, graafikud on tehtud inglise keeles. Töö sisaldab 63 lehekülge teksti, 4 peatükke ja 29 jooniseid.

Information system for restaurants using Microsoft .NET technologies

Annotation

The main purpose of this study is to analyze the problem and implementation of information systems for foodservice.

Diploma thesis consists of two main sections. The first section of this thesis is considered formulation of the problem, the logical structure of data and prototype solutions. The second section examines the technology Windows Presentation Foundation and ASP.NET MVC.

The main result of this work is to implement prototype application for the food service using WPF and the part of the functional (administration) to a separate WEB application.

The work is written in Russian, the application has an English interface, chart made in English. It contains 63 pages of text, 4 chapters, and 29 figures.

Информационная система для ресторанов, используя технологии Microsoft .NET

Аннотация

Основной целью данной работы является анализ задачи и реализация инфосистемы для пункта общественного питания.

Дипломная работа состоит из двух основных разделов. В первом разделе данной дипломной работы рассматривается постановка задачи, логическая структура данных и прототип решения. Во втором разделе рассматривается технология Windows Presentation Foundation и ASP.NET MVC.

Основным результатом данной работы является реализация прототипа приложения для пункта общественного питания с использованием WPF и вынесение части функционала(администрирование) в отдельное WEB приложение.

Работа написана на русском языке, приложение имеет английский интерфейс, диаграммы выполнены на английском языке. Работа состоит из 63 страниц, 4 глав, 29 рисунков.

Оглавление

Терминология.....	7
1. Введение.....	11
1.1. Вступление.....	11
1.2. Проблема.....	12
1.3. Цель работы.....	12
1.4. Рассматриваемая область знаний.....	12
1.5. Структура проекта.....	13
2. Обзор проекта.....	14
2.1. Задачи.....	14
2.2. Действующие лица.....	15
2.3. Структура данных.....	15
2.4. Реализованные функции приложений.....	19
2.5. Структура десктоп приложения.....	20
2.6. Интерфейс десктоп приложения.....	22
2.7. Работа в десктоп приложение.....	26
2.8. Структура web приложения.....	29
2.9. Интерфейс web приложения.....	33
2.10. Работа в web приложение.....	36
3. Обзор технологий.....	38
3.1. Windows Presentation Foundation.....	38
3.2. Entity Framework.....	40
3.3. MVVM.....	44
3.4. MVC.....	48
4. Заключение.....	53
Resümee.....	54
Summary.....	55
Список использованной литературы.....	56
Приложение.....	58

Терминология

.NET Framework – структура программной системы, которая содержит классы, интерфейсы и типы значений, которые облегчают и оптимизируют процесс разработки, а также обеспечивают доступ к функциям системы. Для упрощения взаимодействия между языками большинство типов платформы .NET Framework являются CLS-совместимыми, и поэтому их можно использовать в любом языке программирования, компилятор которого соответствует спецификации CLS. [6,1]

C# (C sharp) – объектно-ориентированный язык программирования. Разработан в 1998 - 2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET.[7,1]

CLS (Common Language Specification) – Подмножество языковых возможностей, которое поддерживается множеством совместимых средств. CLS-совместимые компоненты и средства гарантированно могут взаимодействовать с другими CLS-совместимыми компонентами и средствами [5,1]

CLR (Common Language Runtime) – Общезыковая среда выполнения. Ядро выполнения управляемого кода. Среда выполнения поддерживает управляемый код, обеспечивая его сервисами, такими как, межъязыковая интеграция, безопасность доступа кода, управление жизненным циклом объектов, отладка и профилирование. [5,2]

Data binding (связывание данных) – процесс, который устанавливает соединение между пользовательским интерфейсом приложения и бизнес логикой. Если связывание правильно настроено, то при изменении значений данных, элемент, который связан с данными, автоматически обновляется. [11, 2]

Dependency properties – зависимые свойства. WPF предоставляет набор сервисов, который используется для расширения функциональности CLR свойств. Зависимые свойства предоставляют возможность рассчитывать значение свойства на основе значений других входных данных. [8,1]

SQL (Structured Query Language) – универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. [10,1]

UI (User Interface) – Пользовательский интерфейс. Представляет собой совокупность средств и методов, при помощи которых пользователь взаимодействует с различными машинами и устройствами. [11,1]

WPF (Windows Presentation Foundation) – система следующего поколения для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем.

WPF представляет собой высокоуровневый объектно-ориентированный функциональный слой (framework), позволяющий создавать 2D- и 3D-интерфейсы на базе DirectX [3,2]

XAML (Extensible Application Markup Language) – язык разметки, основанный на XML, который используется для декларативной реализации внешнего вида приложения. Обычно он используется для создания окон, диалоговых окон, страниц и пользовательских элементов управления, а также для их заполнения элементами управления, фигурами и графикой.[3,2]

Шаблон Model-View-ViewModel (**MVVM**) — применяется при проектировании архитектуры приложения. Первоначально был представлен сообществу Джоном Госсманом (John Gossman) в 2005 году

как модификация шаблона Presentation Model. MVVM ориентирован на современные платформы разработки, такие как Windows Presentation Foundation, Silverlight от компании Microsoft, ZK framework.[2, 3]

XML (eXtensible Markup Language) – текстовый формат, предоставляет универсальный способ для описания и обмена структурированной информацией, независимо от приложений и разработчиков, а также является основой для создания более специализированных языков разметки (например, XHTML) [5,4]

ASP.NET(Active Server Pages) — технология создания веб-приложений и веб-сервисов от компании Майкрософт. Она является составной частью платформы Microsoft .NET и развитием более старой технологии Microsoft ASP. На данный момент последней версией этой технологии является ASP.NET 5[1]. ASP.NET внешне во многом сохраняет схожесть с более старой технологией ASP, что позволяет разработчикам относительно легко перейти на ASP.NET. В то же время внутреннее устройство ASP.NET существенно отличается от ASP, поскольку она основана на платформе .NET и, следовательно, использует все новые возможности, предоставляемые этой платформой.[4, 13]

Model-view-controller (MVC, «модель-представление-контроллер», «модель-вид-контроллер») — схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные. [4, 14]

Entity Framework - объектно-ориентированная технология доступа к данным, является object-relational mapping (ORM) решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с

объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL. Технология создана для разделения кода доступа к данным и самих данных. ADO.NET Entity Framework позволяет разработчикам работать с концептуальной моделью данных, в то время как сама модель управляет физической моделью посредством ORM. [1, 1-4]

Синтаксис ASP.NET Razor — это простая модель программирования, позволяющая внедрять серверный код в веб-страницу. Страница также может содержать разметку HTML, данные CSS и клиентские сценарии (JavaScript и jQuery). Синтаксис Razor основан на платформе ASP.NET, которая входит в состав .NET Framework и специально предназначена для создания веб-приложений. Синтаксис Razor предоставляет все возможности ASP.NET, но при этом он проще в изучении для новичков. Если вы эксперт, ваша работа с ним станет продуктивнее. Хотя это простой синтаксис, его связь с ASP.NET означает, что по мере усложнения ваших веб-приложений вы всегда можете воспользоваться возможностями более крупной платформы.[4, 13]

1. Введение

1.1. Вступление

Буквально каждый день в Эстонии открываются новые рестораны, кафе, пекарни, кухни на дому с услугой доставки еды, а также прочие места общественного питания. Все они нуждаются в инфосистеме и соответствующем программном обеспечении позволяющим как минимум:

- Следить за складом
- Оформлять заказы
- Оформлять приём товара
- Составлять регистр продуктов
- Составлять блюда для меню
- Просмотр разного рода статистики
- Проведение инвентур
- Учёт рабочего времени персонала
- Ведение бухгалтерского учёта

Безусловно на рынке программного обеспечения уже существует ни одна фирма предлагающие соответствующий продукт. Главными на рыке в Прибалтике является ЕКТАСО AS и их CompuCashSystem, ближайшем конкурентом является российская Компания R-Keeper. Обе фирмы производят качественное ПО, облегчающий владельцем мест общественного питания видение своего бизнеса.

1.2. Проблема

Открытие даже очень простого ресторана требует больших средств, и поэтому не каждый начинающий предприниматель может позволить себе дорогостоящие программное обеспечение для своего дела. Также многим небольшим заведениям и не нужна часть функционала предоставляющегося программным обеспечением упомянутым выше. Предпринимателю не так важна статистка и учёт рабочего времени персонала, так как, как правило многим занимается он сам, практически 24 часа в сутки, а статистика начинает играть роль только со временем, на старте она очевидна. При этом, и в CompuCashSystem и в R-Keeper, отсутствует возможность удалённого доступа к базе.

1.3. Цель работы

Практической целью работы является реализация инфосистемы и программного обеспечения, которое бы стало эффективным и удобным инструментом для начинающих предпринимателя в сфере общественного питания. Также работа освещает некоторые особенности Microsoft WPF и ASP.NET MVC

1.4. Рассматриваемая область знаний

В данном проекте рассматривается реализация приложения, выполняющего роль инструмента для управления инфосистемой места общественного питания. Приложение реализовано с использованием Microsoft Windows Presentation Foundation и языка C#, базой данных MS SQL, а также часть функционала вынесена в отдельное WEB-приложение на ASP.NET MVC

1.5. Структура проекта

В первой части работы проводится анализ, проектирование и реализация приложения. Ставится проблема, описываются пути решения, определяются задачи, предоставляется реализация.

Вторая часть посвящена особенностям Windows Presentation Foundation и ASP.NET MVC. Некоторые из них рассматриваются на примере использования в реализованном приложении.

2. Обзор проекта

2.1. Задачи

Перед тем как начать разрабатывать приложение необходимо определить основные задачи будущего программного продукта.

1. Реализация операций со складом

- Добавление продуктов в регистр
- Добавление продуктов на склад
- Удаление продуктов из регистра

2. Реализация операций с меню

- Составление блюда из имеющихся продуктов
- Редактирование имеющихся блюд
- Удаление имеющихся блюд

3. Реализация операций с заказами

- Составление заказа
- Редактирование заказа
- Подтверждение заказа

4. Администрирование и просмотр из Онлайн

- Добавление/Удаление пользователей

- Просмотр состояния склада

2.2. Действующие лица

Мы имеем список из двух действующих лиц:

- **Администратор** - имеет доступ ко всем операциям в вышеупомянутом списке задач
- **Пользователь** - имеет доступ ко всем операциям в вышеупомянутом списке задач, кроме пункта **4**

2.3. Структура данных

Вся информация находится в одной базе данных. В качестве хранилища используется *Microsoft SQL Server 2012*. База данных имеет следующие таблицы:

- **Product** - *ProductId*(int, auto_increment, PK), *Name*(nvarchar(50)) - Хранится весь список используемых продуктов.
- **Stock** - *StockId*(int, auto_increment, PK), *Name*(nvarchar(50)) - список складов

- **ProductOnStock** - *ProductOnStockId(int, auto_increment, PK), ProductId(int, FK), Quantity(decimal), BestBefore(dateTime), StockId(int, FK)* - хранение определенного продукта, со сроком годности, с определённым количеством на определённом складе.
- **ProductMenuItem** - *ProductMenuItemId(int, auto_increment, PK), ProductId(int, FK), MenuItemId(Int, FK), Quantity(decimal)* - сколько содержится конкретного продукта в конкретной единице меню(блюде)
- **MenuItem** - *MenuItemId(int, auto_increment, PK), ProductMenuItemId(int, FK), Price(decimal), Name(nvarchar(50)), Active(bit)* - Единица меню(блюдо).
- **OrderRow** - *OrderRowId(int, auto_increment, PK), MenuItemId(int, FK), OrderId(int, FK), Quantity(decimal), Price(decimal)* - Один ряд заказа
- **Order** - *OrderId(int, auto_increment, PK), TotalPrice(decimal), Date(datetime)* - заказ.
- **User** - *UserId(int, auto_increment, PK), UserName(nvarchar(50)), UserPassword(nvarchar(50)), Admin(bit)*

Схемы данных выглядят следующим образом:

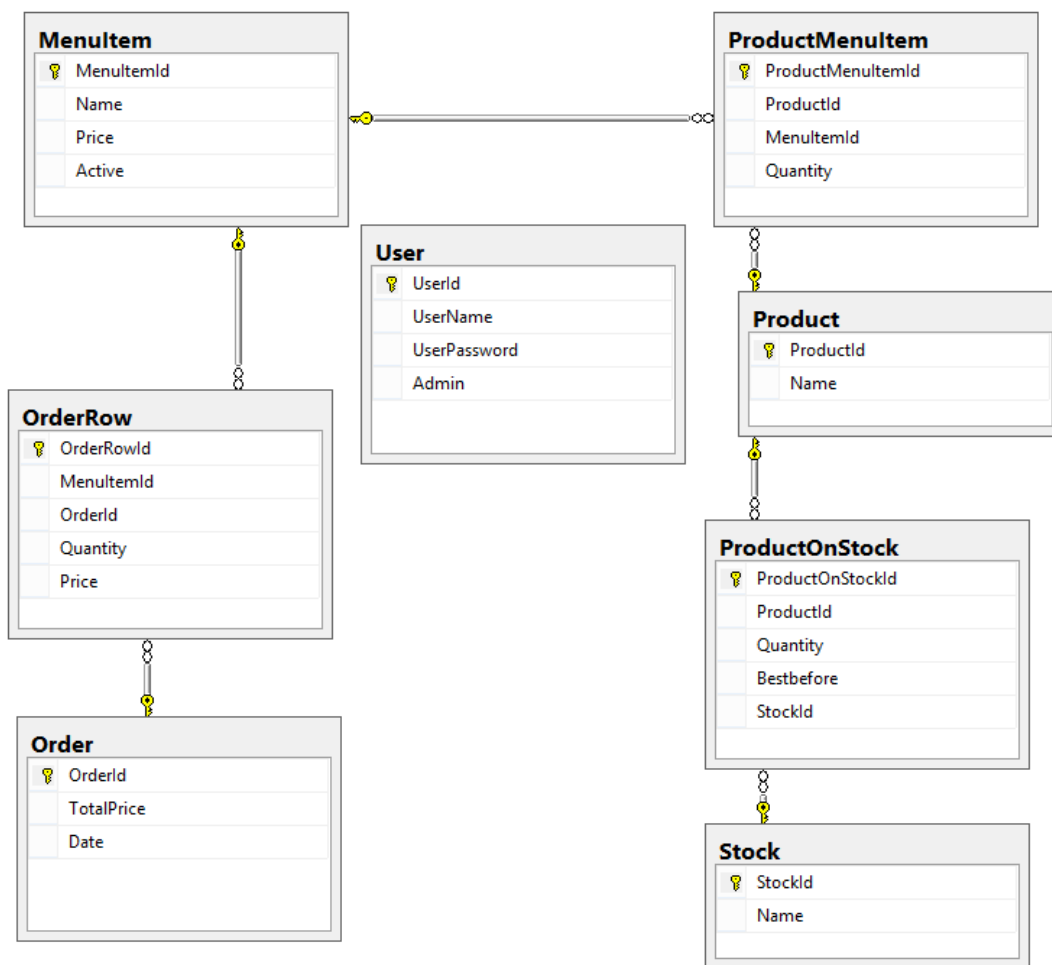


Рисунок 2.3.1. Логическая схема данных

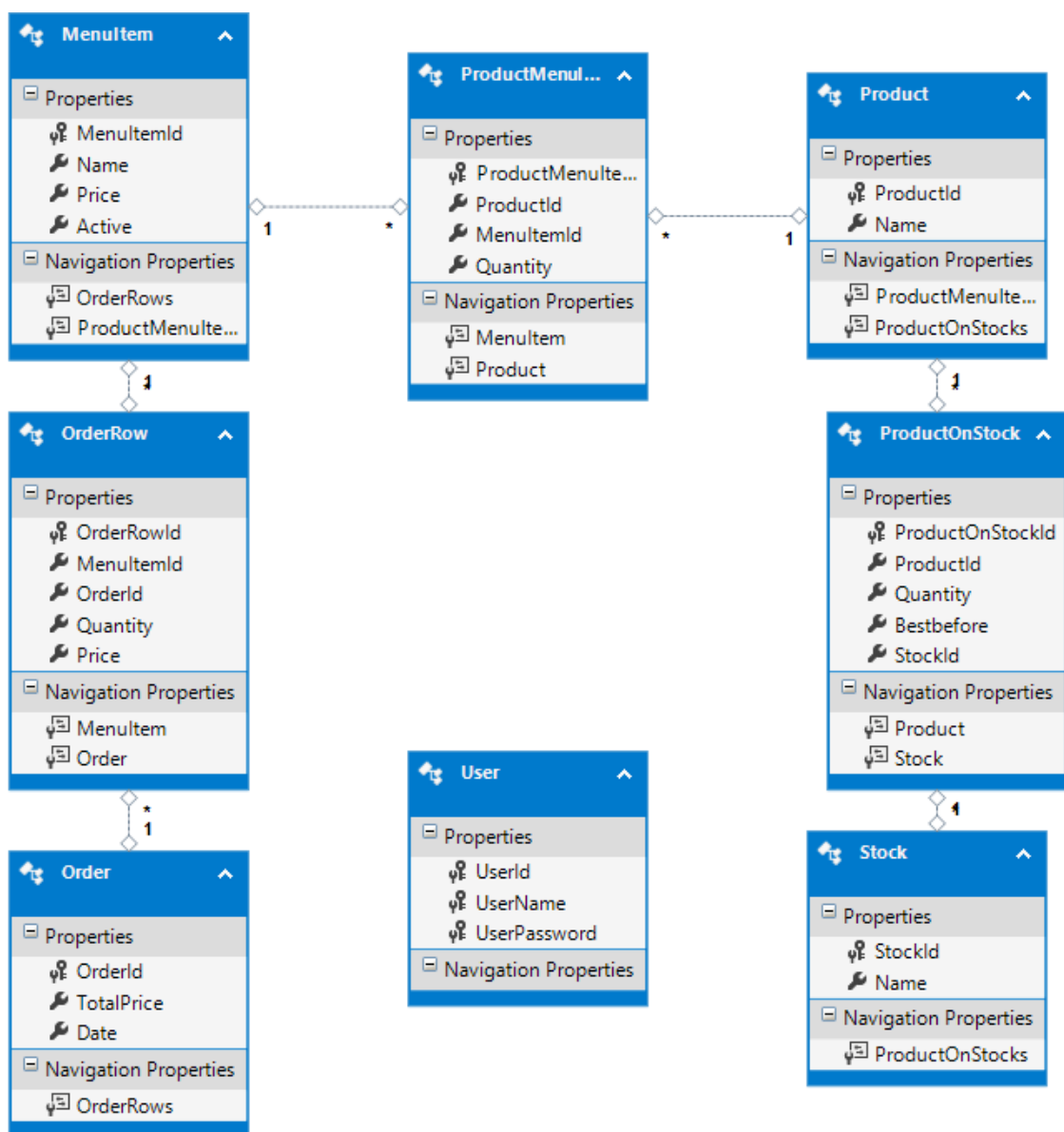


Рисунок 2.3.1. Концептуальная схема данных

2.4. Реализованные функции приложений.

Вывод Информации(Декстоп приложение):

- Список всех продуктов
- Обзор Склада
- Список всех единиц меню(блюд)
- Текущий заказ
- Список составляющих определённой единицы меню

Вывод Информации(Веб приложение):

- Список всех пользователей
- Обзор Склада

Редактирование Информации(Декстоп Приложение)

- Добавление продуктов в регистр
- Добавление продуктов на склад
- Удаление продуктов из регистра
- Составление блюда из имеющихся продуктов
- Редактирование имеющихся блюд
- Удаление имеющихся блюд
- Составление заказа
- Редактирование заказа

Редактирование Информации(Веб Приложение)

- Добавление пользователь
- Удаления пользователя

2.5. Структура десктоп приложения

Проект содержит в себе три каталога: DAL, Model и ViewModel, App.config файл, в котором хранятся настройки проекта, а также LogIn.xaml MainWindow.xaml и MenuItemWindow.xaml, которые являются основными окнами(View) проекта(рис. 2.5.1)

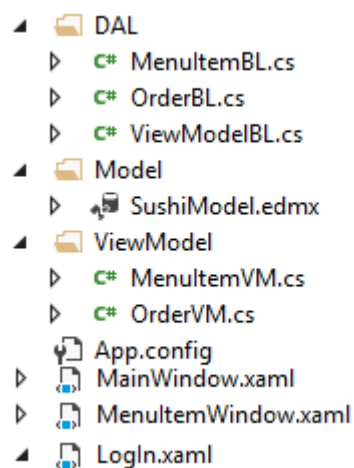


Рисунок 2.5.1. Содержание Десктоп Проекта

Model

Использует технологию *ADO.NET Entity Framework* для составления концептуальной модели и работы с данными.

DAL

Data access layer. Каталог содержит классы доступа к данным(бизнес логика)

ViewModel

В этом каталоге содержатся *Модели представления* для окон(view) приложения

На рисунке 2.5.2. изображена общая структура проекта.

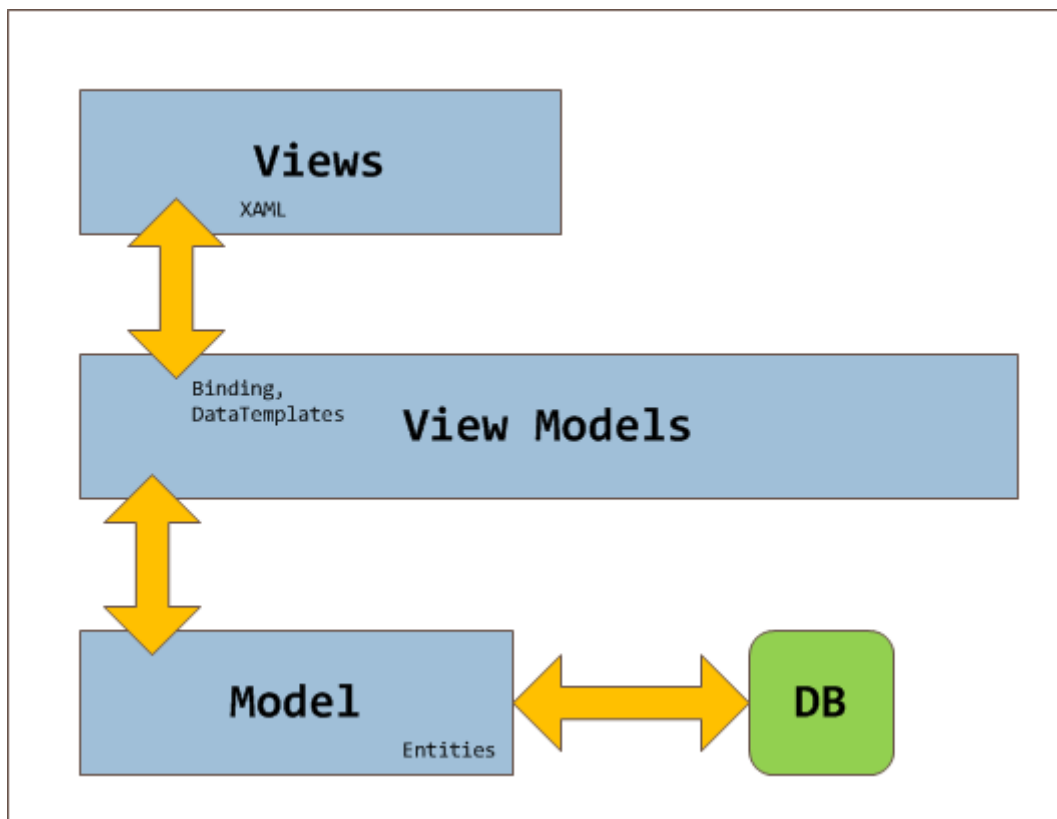


Рисунок 2.5.2. Общая структура Desktop проекта

2.6. Интерфейс десктоп приложения

Интерфейс программы состоит из формы авторизации двух основных форм. Каждая основная форма имеет два вида.

Форма авторизации(рис 2.6.1.)

Имеет два поля ввода для, имя пользователя и пароль пользователя, а также кнопку входа. По нажатию клавиши "Enter" из поля пароль также выполняется вход.

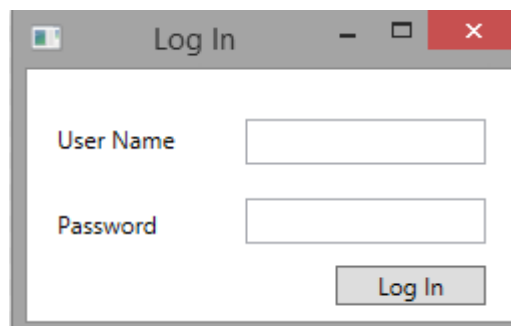


Рисунок 2.6.1. Форма авторизации

Основная форма

- Вид склада(рис. 2.6.2.)

Имеет два списка. Список продуктов с лева и список продуктов на складе с права. Также имеется кнопка добавления продукта на склад и удаления продукта из регистра. В нижней части формы можно увидеть поле ввода и кнопку, реализованных для добавления новых продуктов в регистр.

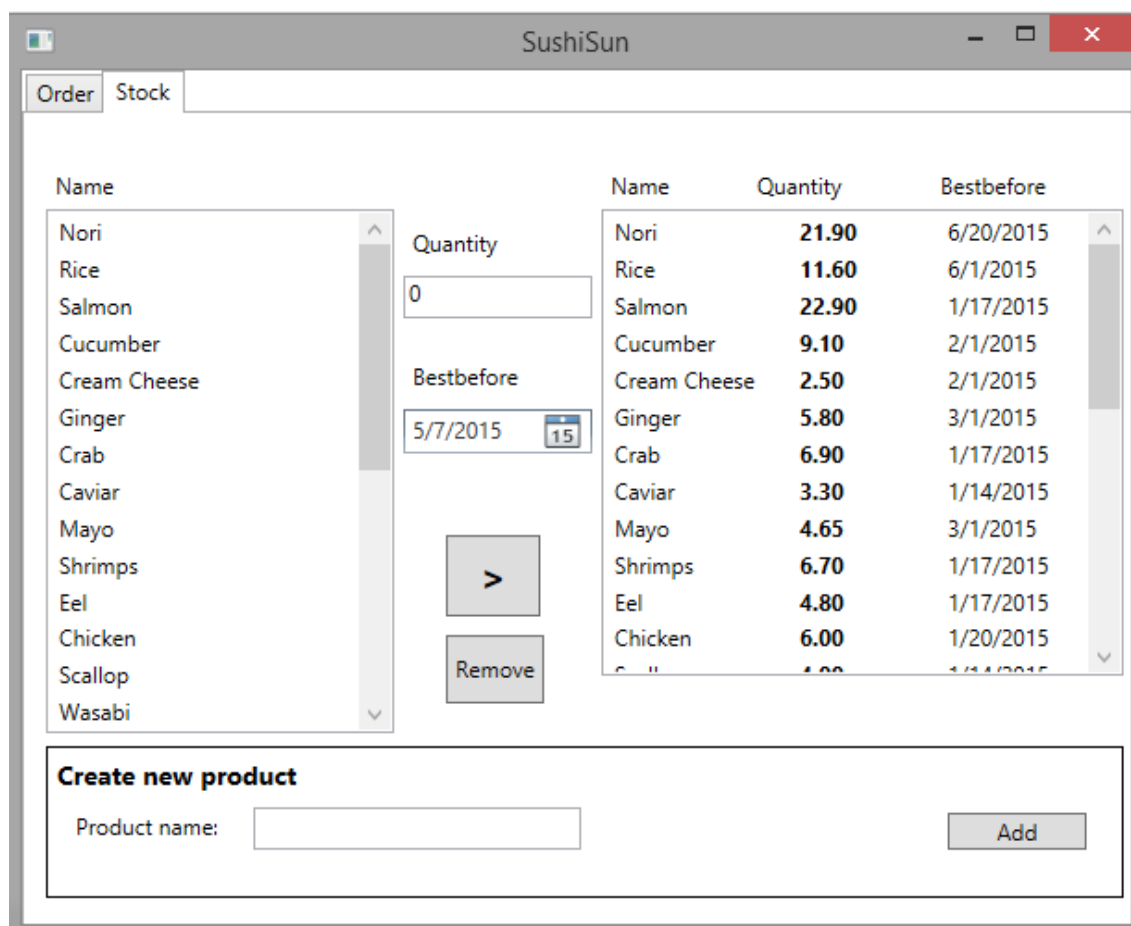


Рисунок 2.6.2. Вид Склада

- Вид Заказа(рис. 2.6.3.)

Имеет два списка. С права список всех единиц меню, с лева вид текущего заказа. Две кнопки, добавить единицу меню в заказ и соответственно убрать единицу меню из заказа. Под списком текущего заказа имеется значение суммы всего заказа и кнопка подтверждения заказа. Под списком меню имеется кнопка удаление единицы меню и добавления единицы меню

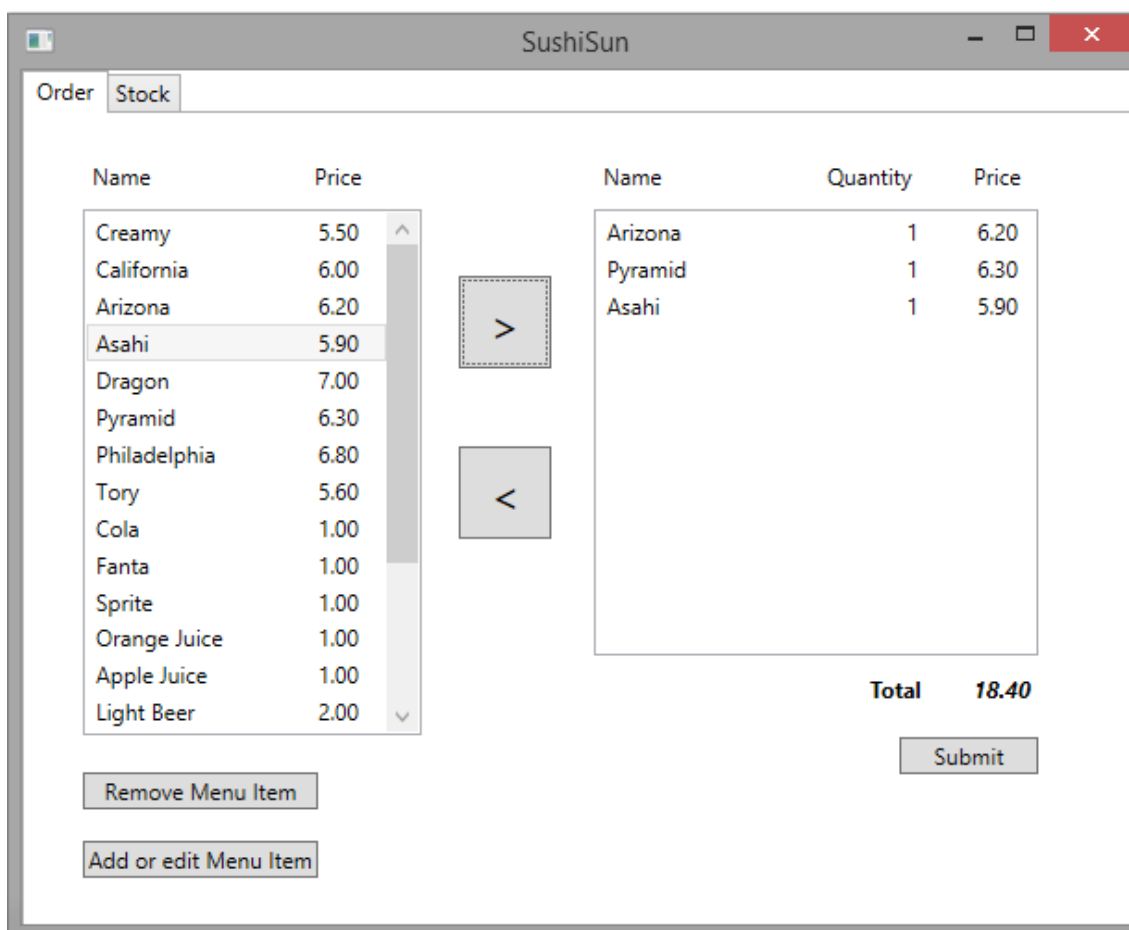


Рисунок 2.6.3. Вид заказа

Форма единицы меню

- Вид создания единицы меню(рис. 2.6.4)

Имеется два списка. С левой стороны список всех продуктов, с правой страны список уже добавленных продуктов. Посередине есть поля две кнопки, у брать или добавить определённый продукт, в определённом количестве. С низу имеется ещё два поля ввода, для введения названия нового блюда и его цены.

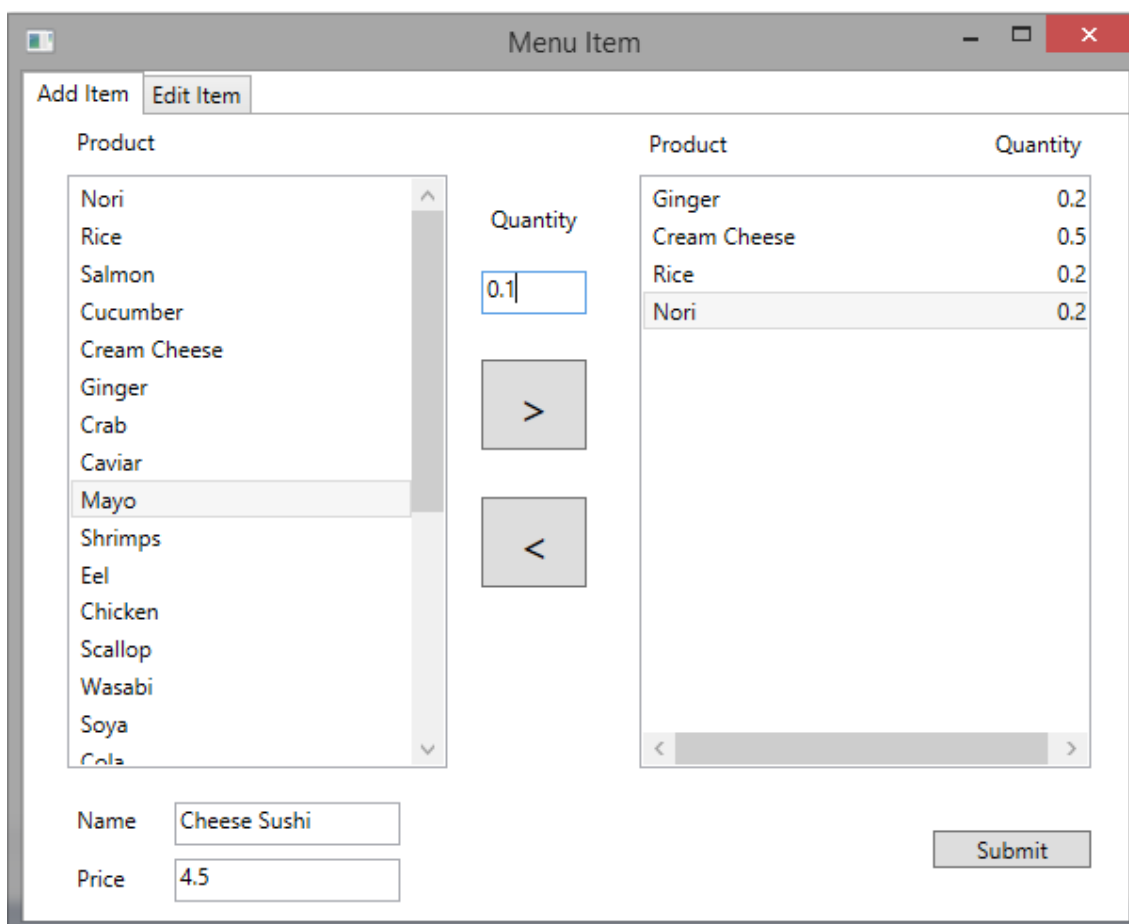
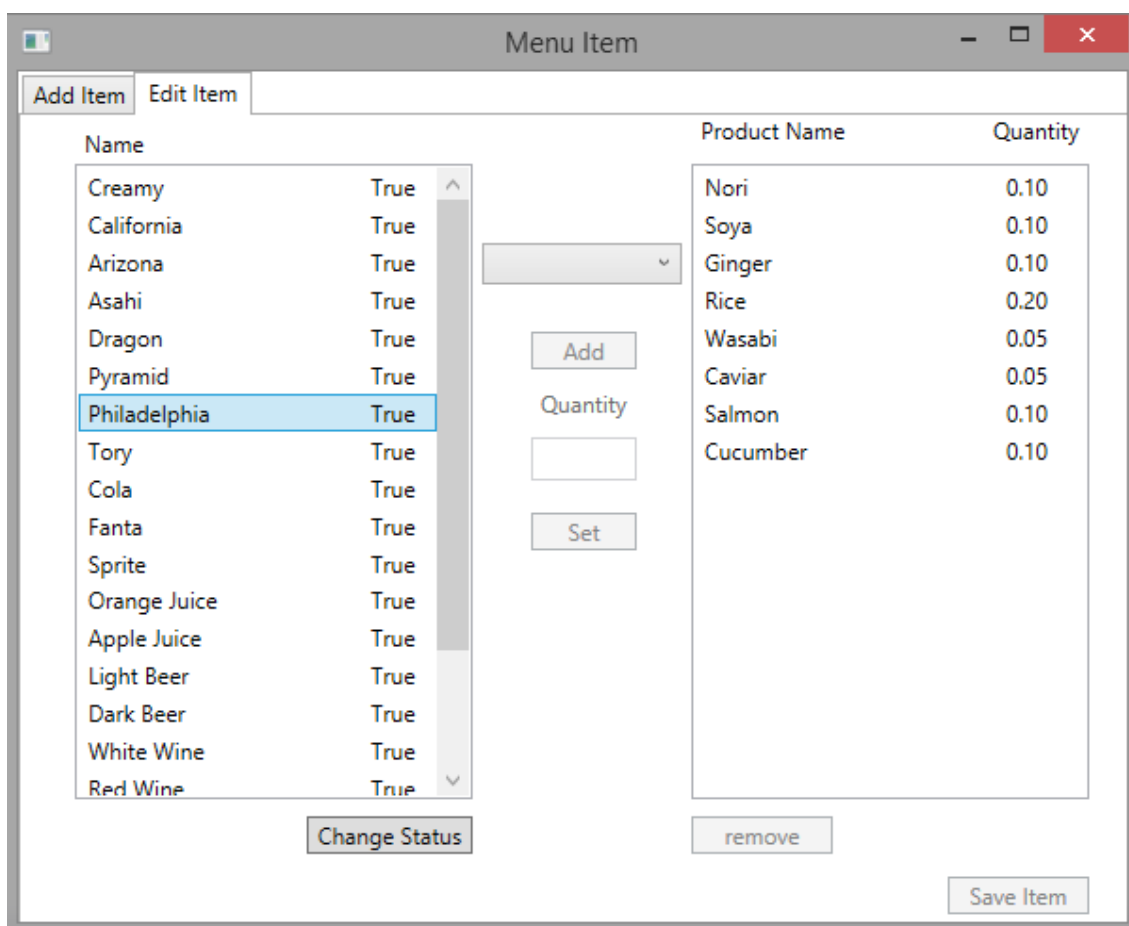


Рисунок 2.6.4. Вид создания единицы меню

- **Вид редактирования единицы меню(рис. 2.6.5)**

Имеется два списка. С левой стороны список единиц меню, с правой стороны, список составляющих ее продуктов. Посередине есть "выпадающий" список продуктов, а также кнопка "добавить", и кнопка "выставить". Снизу под списком продуктов, есть кнопка "убрать" и кнопка "сохранить".



2.7. Работа в десктоп приложение

- **Авторизация**

При запуски приложения первым появляется окно авторизации. Следует ввести имя пользователя и соответствующий пароль, далее вы попадёте в основное окно приложения.

- **Заполнение регистра продуктов**

Эта операция непосредственна необходима, для всех последующих действий, но несмотря на это, она не находится на основной вкладке, потому что нам надо заполнить регистр только раз, а потом лишь редактировать его при надобности.

Переходим с основной вкладки на вкладку "Склада", в поле ввода печатаем название продукта и соответственно кнопкой или клавишей "Ввод" с поля ввода добавляем продукт. Кнопка "Убрать" уберёт выбранный продукт.

- **Заполнения складов**

Теперь непосредственно можно заняться заполнением склада. Выбираем продукт из регистра, выставляем количество в поле ввода, и срок годности в поле выбора даты. Нажимаем на кнопку ">" и теперь мы имеем на складе определённый продукт, в определённом количестве с определённой датой срока годности. Кнопка "Убрать" уберёт выбранный продукт со склада, при ошибочном вводе или списывание продукта.

- **Составления единиц меню**

Эта операция непосредственна необходима, для последующих действий, но она не находится на основной вкладке, по той же причине, что и операция заполнения регистра продуктов.

Кнопкой "Добавить или Редактировать единицу меню" переходим в новое окно. Из списка продуктов выбираем нужный нам продукт, выставляем его количество и добавляем его кнопкой ">", при ошибочном вводе можно убрать продукт кнопкой "<". При повторном добавление продукта, новый ряд не создаётся, а просто увеличивается количество уже имеющегося продукта. Вводим в поле название новой единицы меню и его

цену, и нажимаем кнопку "Подтвердить". Также на этой вкладке имеется кнопка восстановления удалённой единицы меню.

- **Редактирование единиц меню**

Эта операция предназначена для исправления ошибок допущенных при составление единицы блюда или просто для изменения его рецептуры. Выбираем единицу меню, с права отображается её содержание. Выбирая продукт слева, можно изменить его количество в данной единице меню. Также в "выпадающем" списке содержатся все продукты, выбрав один из них, выставив его количество в поле ввода, и нажав кнопку "выставить", мы добавим новый продукт в рецепт. Кнопка "убрать", соответственно убирает выделенные продукт из рецепта. Для сохранения новой рецептуры следует нажать соответствующую кнопку.

- **Составление заказов**

Выделаешь в списке единиц меню нужное тебе блюдо или напиток и нажимаешь кнопку ">" при повторном нажатии, увеличивается количество в заказе. Если надо убрать или изменить количество, то можно воспользоваться кнопкой "<", когда заказ составлен, следует нажать кнопку подтверждения, при её нажатие, со склада автоматически списывается количество продуктов задействованных в заказе.

2.8. Структура web приложения

Проект содержит в себе такие каталоги, как: App_Data, App_Start, Content, fonts, Scripts, все они сгенерированы Visual Studio и содержат необходимые данные для дальнейшей работы с front-end - ом. В Рамках нашей работы нас больше интересует back-end составляющая, поэтому мы рассмотрим такие каталоги как: Controllers, Models, Views (Рис. 2.8.1)

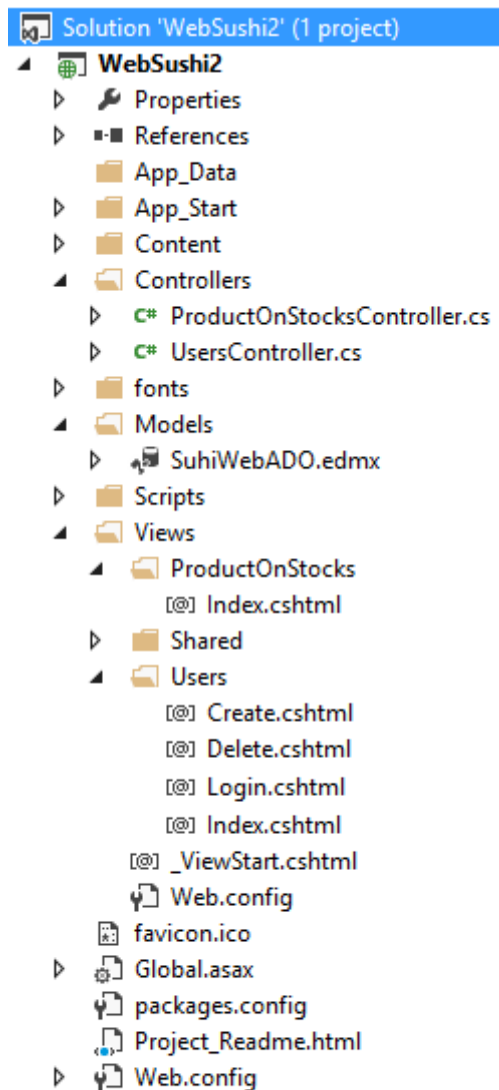


Рисунок 2.8.1 Содержание web Проекта

Model

Использует технологию *ADO.NET Entity Framework* для составления концептуальной модели и работы с данными. В данном проекте для

работы, мы имплементировали только необходимые нам таблицы. Products, ProductsOnStock, User.

Controllers

В данном каталоге содержатся два контролера, для работы с пользователями и для работы со складом.

Views

Виды нашего приложения. Как видно из рисунка, склад имеет только основной вид(index), в то время как у пользователя есть ещё три дополнительных вида: Login, Create, Delete.

На рисунке 2.8.2. изображена общая структура проекта.

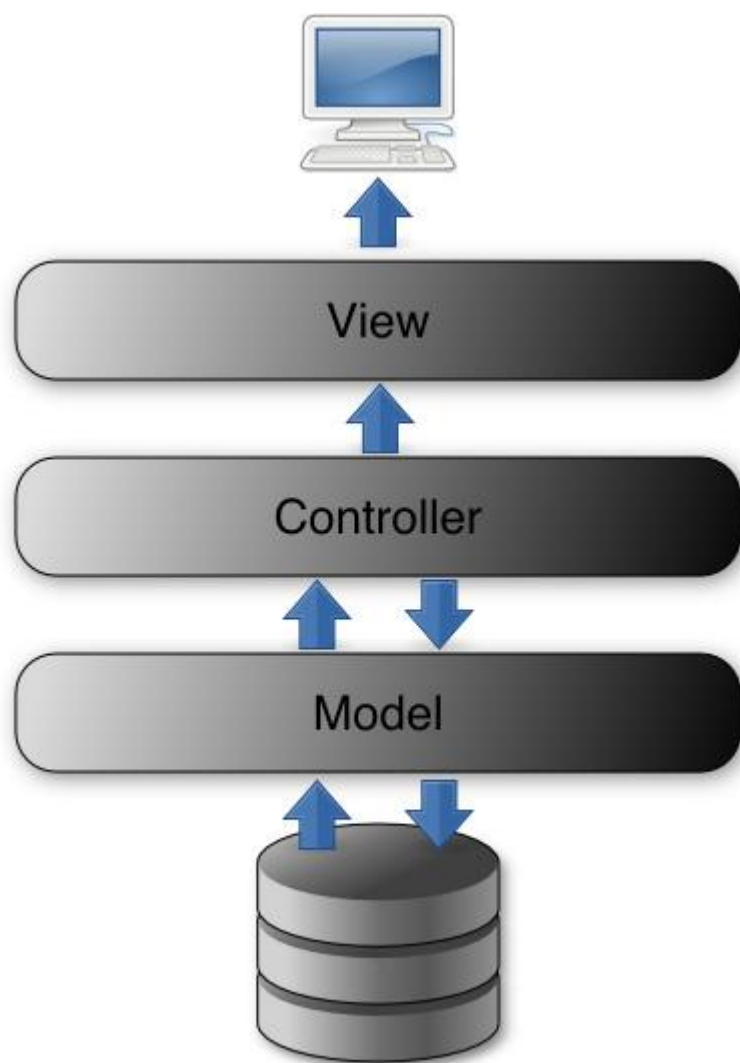


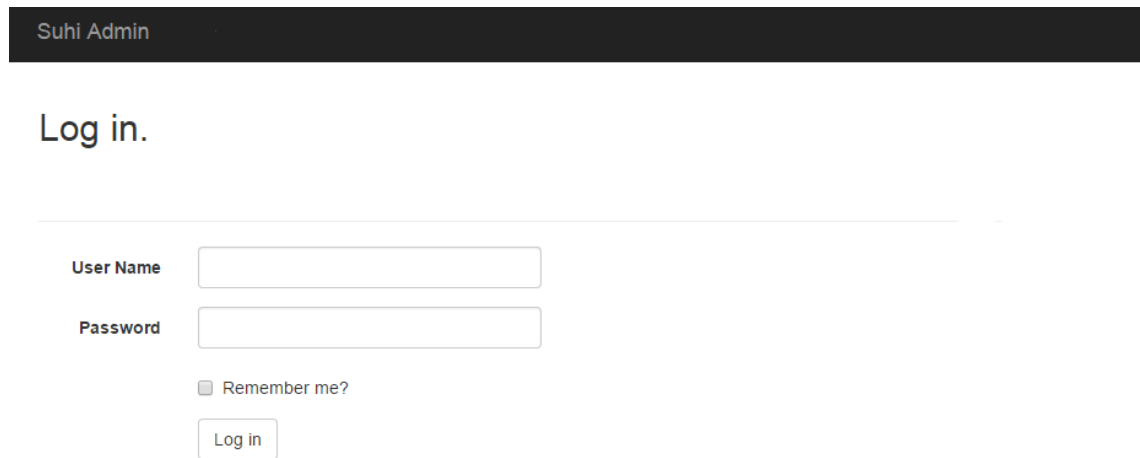
Рисунок 2.8.2. Общая структура web проекта

2.9. Интерфейс web приложения

Интерфейс web - приложения составляет один основной вид склада, а также основной вид пользователей(лист пользователей), а также вид авторизации, вид добавление и удаления пользователей.

- **Вид Авторизации(рис. 2.9.1.)**

Представляет из себя вид страничку, с двумя полями ввода, полем для выставления галочки "Запомнить пользователя" и кнопкой "Log In".



Suhi Admin

Log in.

User Name

Password

Remember me?

Рисунок 2.9.1. Вид Авторизации WEB

- **Вид Склада(рис. 2.9.2.)**

Представляет из себя таблицу продуктов. Имеет три столбца: Название продукта, количество и дату(срок годности). Отображает состояние склада на данный момент.

Store

Name	Quantity	Bestbefore
Apple Juice	20.00	01 Jun 2015
Caviar	3.30	14 Jan 2015
Chicken	6.00	20 Jan 2015
Cola	39.50	01 Sep 2015
Crab	6.90	17 Jan 2015
Cream Cheese	2.50	01 Feb 2015
Cucumber	9.10	01 Feb 2015
Dark Beer	20.00	01 Jun 2015
Eel	4.80	17 Jan 2015
Fanta	38.50	01 Sep 2015
Ginger	5.80	01 Mar 2015
Light Beer	17.50	01 Jun 2015
Mayo	4.65	01 Mar 2015
Nori	21.90	20 Jun 2015
Nori	3.90	30 Jan 2015
Orange Juice	19.00	01 Jun 2015
Red Wine	14.50	01 May 2015
Rice	11.60	01 Jun 2015
Salmon	22.90	17 Jan 2015
Salmon	1.00	18 Jan 2015
Scallop	4.00	14 Jan 2015
Shrimps	6.70	17 Jan 2015

Рисунок 2.9.2. Вид склада WEB

- **Вид Пользователей(рис. 2.9.3.)**

Страничка, отображающая список пользователей. Представляет из себя таблицу, со столбцами имён и полям для галочки, которая указывает на то, является ли данный пользователь администратором. Также имеется две гиперссылки, для создания нового пользователя("Create New", над таблицей) и удаления конкретного пользователя в таблице.

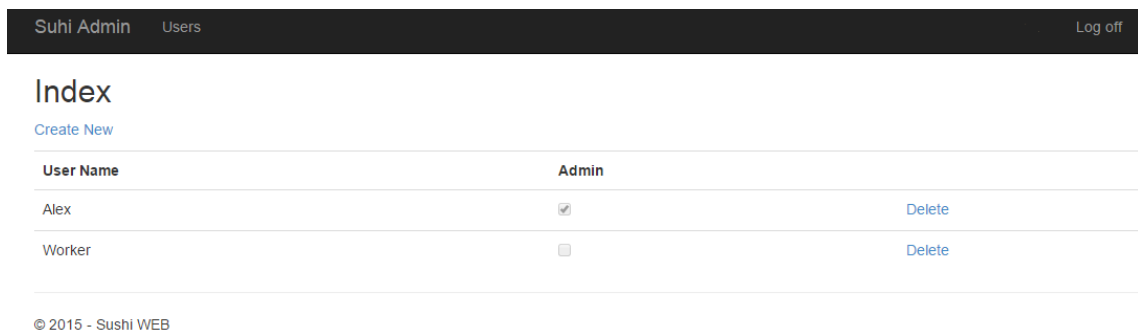


Рисунок 2.9.4. Пользователи WEB

- **Вид добавления нового пользователя(рис. 2.9.5.)**

Представляет из себя страничку, с двумя полями ввода и полем для выставления галочки, которая указывает на то, является ли будущий пользователь администратором. Также имеется кнопка "Создать" и гиперссылка "Вернуться к листу пользователей"

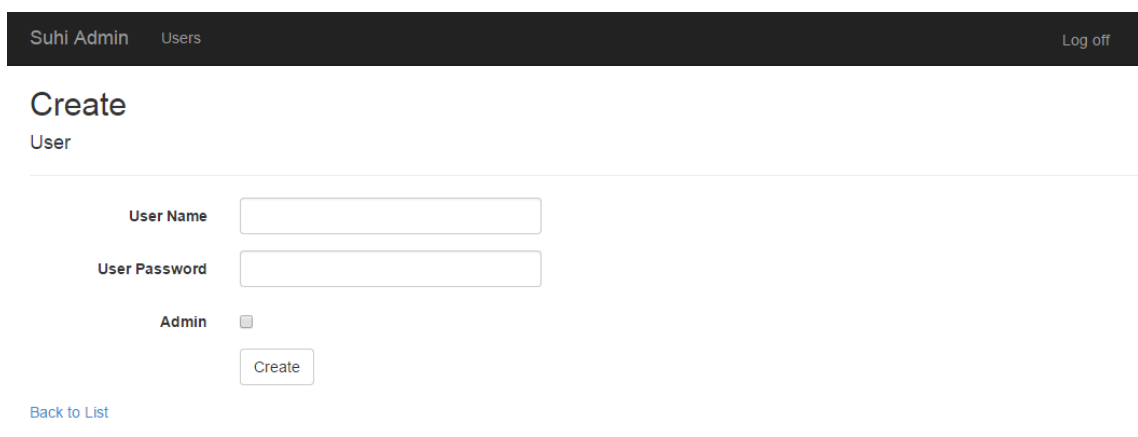


Рисунок 2.9.5. Добавление Пользователя WEB

- **Удаление пользователя(рис. 2.9.6.)**

Страничка с текстом имени пользователя, кнопкой удалить и гиперссылкой "Вернуться к листу пользователей"

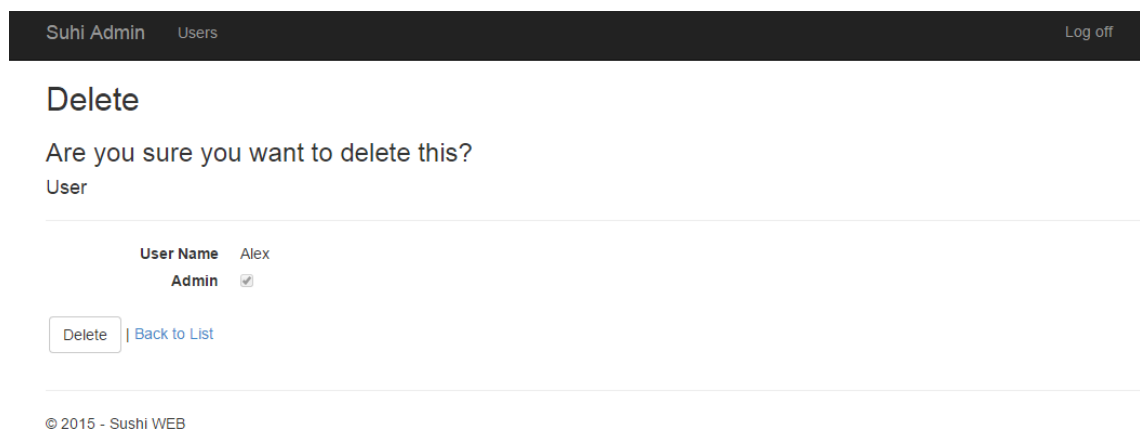


Рисунок 2.9.5. Удаление Пользователя WEB

2.10. Работа в web приложение

- **Авторизация**

При заходе на страницу приложения, первым делом надо провести авторизацию. Следует ввести имя пользователя и соответствующий пароль. Также, пользователь должен быть администратором, чтобы авторизация была успешной.

- **Просмотр склада**

Главным окном является просмотр склада. Администратор может наблюдать состояние склада в данный момент.

- **Просмотр пользователей**

Перейдя по ссылке "Пользователи" в верхней части экрана, администратор попадает на вид листа пользователей. С этой страницы можно перейти по гиперссылке на страницу удаления или добавления пользователя

- **Удаление пользователя**

После нажатие на гиперссылку "удалить" вы попадаете на страницу удаления пользователя, где следует подтвердить свои намерения об удалении, либо, при ошибочном клике, можно вернуться назад к списку пользователей

- **Добавление пользователя**

Вводим в поле ввода имя пользователя и его пароль, при необходимости создания пользователя-администратора, ставим соответственно галочку в поле "Администратор", нажимаем кнопку "Создать"

3. Обзор технологий

3.1. Windows Presentation Foundation

Windows Presentation Foundation (WPF) — это система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем. С помощью WPF можно создавать широкий спектр приложений.

WPF расширяет базовую систему полным набором функций разработки приложений, в том числе Язык XAML (Extensible Application Markup Language), элементами управления, привязкой данных, макетом, двухмерной- и трехмерной-графикой, анимацией, стилями, шаблонами, документами, мультимедиа, текстом и оформлением. WPF входит в состав Microsoft .NET Framework и позволяет создавать приложения, включающие другие элементы библиотеки классов .NET Framework. Основными особенностями WPF являются:

- **Векторная визуализация**

Основной единицей измерения в графической системе WPF является аппаратно-независимый пиксель, который составляет 1/96 часть дюйма независимо от фактического разрешения экрана и предоставляет основу для создания изображения, независимого от разрешения и устройства. Каждый аппаратно-независимый пиксель автоматически масштабируется в соответствии с числом точек на дюйм в системе, в которой он отображается. Графической технологией, лежащей в основе WPF, является DirectX.

Так как WPF использует векторную визуализацию, то это позволяет использовать преимущества мониторов с любым разрешением, без каких либо дополнительных усилий разработчика или пользователя. Интерфейс пользователя перестает зависеть от данных настроек компьютера.

- **XAML**

Или eXtensible Application Markup Language является языком разметки для приложений, основанный на XML язык разметки, разработанный Microsoft.

В WPF, XAML определяет внешний вид приложения, при этом он не так тесно связан с кодом поведения приложения, что позволяет параллельно разрабатывать дизайн и back-end приложения. По своей структуре и синтаксису, XAML чем-то напоминает HTML(рис 3.1.1.)

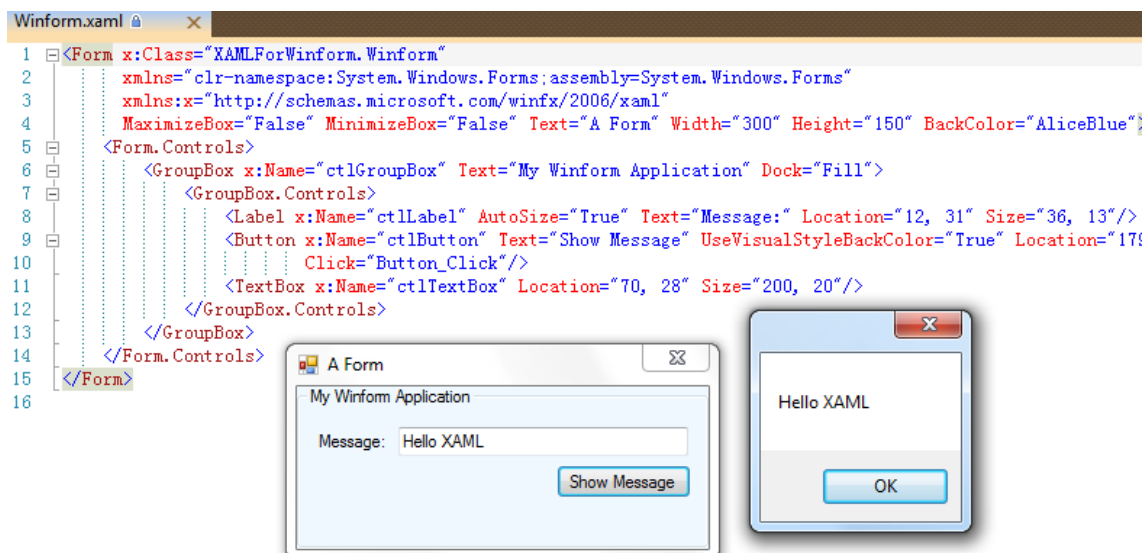


Рисунок 3.1.1. Пример простого XAML приложения

- **Анимация**

В WPF нет необходимости использовать таймер для того, чтобы заставить форму перерисовать себя, как это было необходимо делать в Windows Forms. Вместо этого доступна анимация — неотъемлемая часть платформы. Анимация определяется декларативными дескрипторами, и WPF запускает ее в действие автоматически.

3.2. Entity Framework

Объектно-реляционный модуль сопоставления, позволяющий разработчикам .NET работать с реляционными данными с помощью объектов, специализированных для доменов. Это устраняет необходимость в написании большей части кода для доступа к данным, который обычно требуется разработчикам. В данном проекте мы

использовали Entity Framework, чтобы создать модель данных. Данная модель позволяет создать программируемые классы, с помощью которых можно быстрее и удобнее манипулировать данными в базе данных.

Существует 2 подхода создания модели данных в приложении.

- **Database-first design** - сначала создается база данных, впоследствии генерируется модель при помощи *Entity Framework* исходя из таблиц базы данных.
- **Model-first design** – создается концептуальная модель, исходя из которой позднее строится база данных. В данном случае разработчик описывает модель исходя из бизнес-процессов, которые необходимо реализовать. Структура базы данных строится для поддержания этой концепции.

В рамках нашего проекта, для создания модели, мы использовали **Database-first(рис. 3.2.1)**, поскольку оба проекта используют одно и тоже хранилище данных, с той лишь разницей, что десктоп приложение использует все сущности, а web ограничивается лишь необходимой для администрирования частью.

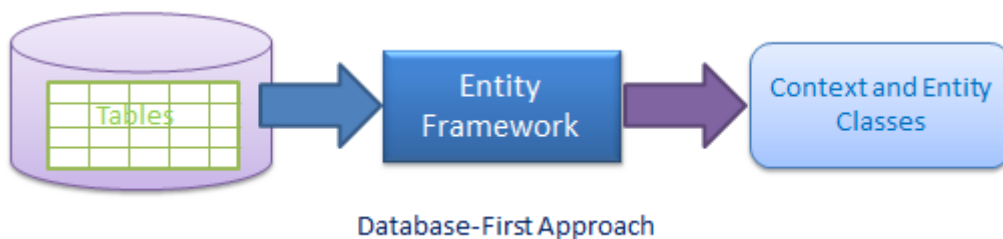


Рисунок 3.2.1. Database-first

Создав модель, мы получили нужные нам классы, для дальнейшей работы с базой данных. Непосредственно класс SushiDBEntities(рис. 3.2.2.), унаследованный от DbContext класса, который позволяет нам дальнейшую работу с базой.

```
public partial class SushiDBEntities : DbContext
{
    public SushiDBEntities1()
        : base("name=SushiDBEntities")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    public virtual DbSet<MenuItem> MenuItem { get; set; }
    public virtual DbSet<Order> Orders { get; set; }
    public virtual DbSet<OrderRow> OrderRows { get; set; }
    public virtual DbSet<Product> Products { get; set; }
    public virtual DbSet<ProductMenuItem> ProductMenuItems { get; set; }
    public virtual DbSet<ProductOnStock> ProductOnStocks { get; set; }
    public virtual DbSet<User> Users { get; set; }
    public virtual DbSet<Stock> Stocks { get; set; }
}
```

Рисунок 3.2.2. Класс SushiDBEntities

Также, мы получили, все классы сущностей, которые представляют из себя модели конкретных таблиц. Пример, сущность продукта(рис. 3.2.3.).

```

namespace SushiSun.Model
{
    using System;
    using System.Collections.Generic;

    public partial class Product
    {
        public Product()
        {
            this.ProductMenuItems = new HashSet<ProductMenuItem>();
            this.ProductOnStocks = new HashSet<ProductOnStock>();
        }

        public int ProductId { get; set; }
        public string Name { get; set; }

        public virtual ICollection<ProductMenuItem> ProductMenuItems { get; set; }
        public virtual ICollection<ProductOnStock> ProductOnStocks { get; set; }
    }
}

```

Рисунок 3.2.3. Класс *Product*

Сгенерировав модель, у нас появилась дальнейшая возможность работы с базой. Для запросов к базе использовали **LINQ to Entities(рис. 3.2.4.)** технологию, которая позволяет обращаться к хранилищу данных используя *Language Integrated Query(LINQ)*. Например, функция получения сущности продукта на складе с конкретным Id выглядит таким образом.

```

public ProductOnStock getProductOnStockkByProductId(int id)
{
    ProductOnStock product;
    using (SushiDBEntities db = new SushiDBEntities())
    {
        product = db.ProductOnStocks.OrderBy(x => x.Bestbefore).Where(x => x.ProductId == id).FirstOrDefault();
    }
    return product;
}

```

Рисунок 3.2.4. *LINQ to Entities*

3.3. MVVM

Model-View-ViewModel удобно использовать вместо классического MVC и ему подобных в тех случаях, когда в платформе, на которой ведётся разработка, присутствует «связывание данных»

Шаблон MVVM делится на три части(рис.3.3.1.):

- **Модель** (Model) представляет собой фундаментальные данные, необходимые для работы приложения. В рамках данного проекта, моделями являются сущности(Entities), полученные при использовании Entity Framework.
- **Представление** (View) — это графический интерфейс, то есть окно, кнопки и т. п. В данном проекте реализованы с помощью XAML. Представление является подписчиком на событие изменения значений свойств или команд, предоставляемых Моделью представления. В случае, если в Модели представления изменилось какое-либо свойство, то она оповещает всех подписчиков об этом, и Представление, в свою очередь, запрашивает обновленное значение свойства из Модели представления. В случае, если пользователь воздействует на какой-либо элемент интерфейса, Представление вызывает соответствующую команду, предоставленную Моделью представления.
- **Модель представления** (ViewModel) является, с одной стороны, абстракцией Представления, а с другой, предоставляет обёртку

данных из Модели, которые подлежат связыванию. То есть, она содержит Модель, которая преобразована к Представлению, а также содержит в себе команды, которыми может пользоваться Представление, чтобы влиять на Модель.

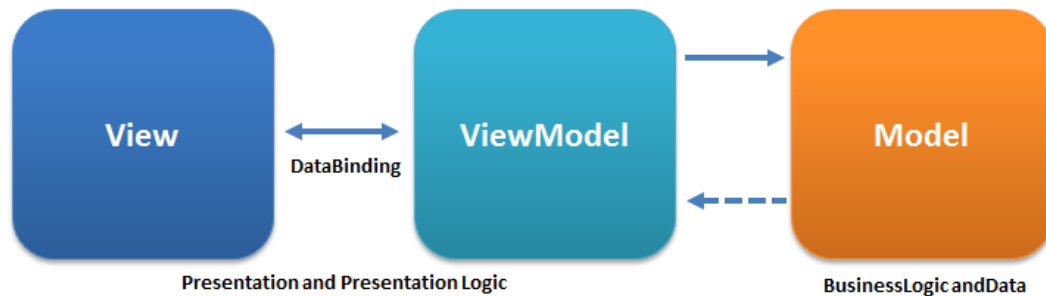


Рисунок 3.3.1. MVVM схема

В рамках данного проекта MVVM используется в десктоп приложении. Рассмотрим на примере процесса редактирования единицы меню. Мы имеем ViewModel класс(рис. 3.3.2.), который содержит в себе коллекции для отображения, редактирования и дальнейшего сохранения, соответствующие свойства для привязке к представлению, а также методы для работы с данными и конструктор.

```

public class MenuItemVM
{
    #region private fields

    private ViewModelBL _viewModelBL;
    private ObservableCollection<Product> _products;
    private ObservableCollection<ProductMenuItem> _productMenuItems;
    private ObservableCollection<MenuItem> _menuItems;
    private ObservableCollection<ProductMenuItem> _productMenuItemsEdit;

    #endregion

    #region public propirties

    public ObservableCollection<Product> Products
    {
        get { return _products; }
    }
    public ObservableCollection<ProductMenuItem> ProductMenuItems
    {
        get { return _productMenuItems; }
        set { _productMenuItems = value; }
    }

    public ObservableCollection<MenuItem> MenuItems
    {
        get { return _menuItems; }
        set { _menuItems = value; }
    }
    public ObservableCollection<ProductMenuItem> ProductMenuItemsEdit
    {
        get { return _productMenuItemsEdit; }
        set { _productMenuItemsEdit = value; }
    }

    #endregion

    constructors

    methods
}

```

Рисунок 3.3.2. Класс MenuItemVM

В соответствующем коде интерфейса производится привязка данных, к соответствующим полям(рис. 3.3.3.).

```
<ListBox x:Name="lbEditMenuItem"
    HorizontalAlignment="Left"
    Height="337" Margin="27,24,0,0"
    VerticalAlignment="Top"
    Width="211"
    ItemsSource="{Binding MenuItems}" SelectionChanged="lbEditMenuItem_SelectionChanged">
<ListBox.ItemTemplate>
    <DataTemplate>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="150"></ColumnDefinition>
                <ColumnDefinition Width="*"></ColumnDefinition>
            </Grid.ColumnDefinitions>
            <TextBlock TextAlignment="Left" Text="{Binding Name}" Grid.Column="0"></TextBlock>
            <TextBlock TextAlignment="Right" Text="{Binding Active}" Grid.Column="1"></TextBlock>
        </Grid>
    </DataTemplate>
</ListBox.ItemTemplate>
</ListBox>
```

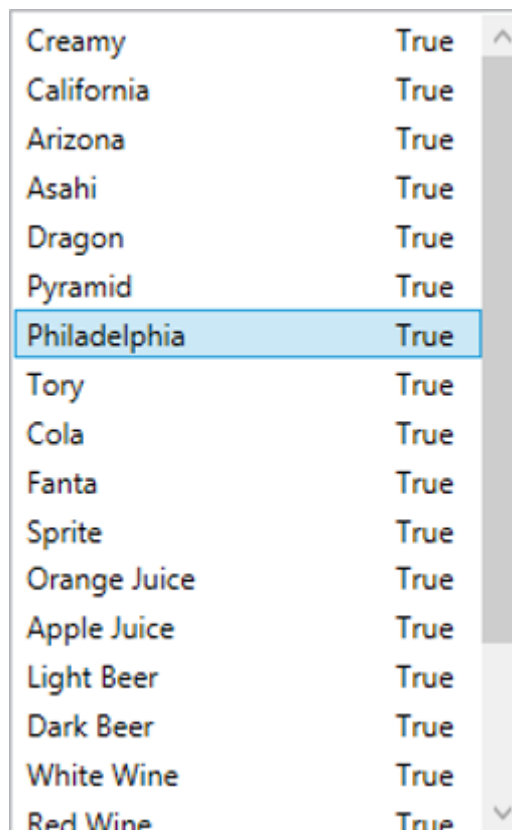
Рисунок 3.3.3. Привязка данных

Таким образом сам список у нас привязывается к свойству MenuItemList нашего ViewModel класса, а два столбца этого списка привязываются к свойствам класса MenuItemList(Name, Active). При инициализации данного интерфейса следует указать класс ViewModel, как источник данных данного окна(рис. 3.3.4.).

```
public MenuItemWindow()
{
    InitializeComponent();
    _menuItemVM = new MenuItemVM();
    _menuItemBL = new MenuItemBL();
    _menuItemVM.GetData();
    this.DataContext = _menuItemVM;
}
```

Рисунок 3.3.4. Инициализация интерфейса

Теперь при запуске данного интерфейса мы получим лист всех единиц меню(рис.3.3.5).



The image shows a screenshot of a menu unit list. The list contains 16 items, each with a name and a 'True' status. The 'Philadelphia' item is highlighted with a blue selection bar. The list is contained within a scrollable window with a vertical scrollbar on the right side.

Creamy	True
California	True
Arizona	True
Asahi	True
Dragon	True
Pyramid	True
Philadelphia	True
Tory	True
Cola	True
Fanta	True
Sprite	True
Orange Juice	True
Apple Juice	True
Light Beer	True
Dark Beer	True
White Wine	True
Red Wine	True

Рисунок 3.3.5. Список единиц меню

3.4. MVC

MVC представляет собой стандартный шаблон разработки. В состав платформы MVC входят следующие компоненты.

- **Модели.** Объекты моделей являются частями приложения, реализующими логику для домена данных приложения. Объекты моделей часто получают и сохраняют состояние модели в базе данных.

- **Представления.** Представления служат для отображения пользовательского интерфейса приложения. Пользовательский интерфейс обычно создается на основе данных модели.
- **Контроллеры.** Контроллеры осуществляют взаимодействие с пользователем, работу с моделью, а также выбор представления, отображающего пользовательский интерфейс. В приложении MVC представления только отображают данные, а контроллер обрабатывает вводимые данные и отвечает на действия пользователя. Например, контроллер может обрабатывать строковые значения запроса и передавать их в модель, которая может использовать эти значения для отправки запроса в базу данных.

В рамках данного проекта MVC использовалась для реализации web-приложения. Рассмотрим данную технологию подробнее на примере процесса «Создать пользователя».

В качестве модели использовалась сущность «Пользователь» созданная с помощью Entity Framework.(рис. 3.4.1.) и дополненная необходимыми аннотациями.

```
public partial class User
{
    public int UserID { get; set; }
    [Display(Name = "User Name")]
    public string UserName { get; set; }
    [DataType(DataType.Password)]
    [Display(Name = "User Password")]
    public string UserPassword { get; set; }
    public bool Admin { get; set; }
}
```

Рисунок 3.4.1. Модель Пользователь

Представлением является динамическая веб страничка с использованием синтаксиса ASP.NET Razor(рис 3.4.2.). При выполнении страницы с кодом Razor сервер выполняет код перед отправкой страницы в браузер. Выполнение кода на сервере позволяет выполнять более сложные задачи, чем допускает клиент, например работать с серверными базами данных. Более того, серверный код позволяет динамически генерировать клиентский контент. Он может создавать разметку HTML или другой контент, чтобы отправить его в браузер вместе со статическим HTML на странице.

```

@model WebSushi2.Models.User
@{
    ViewBag.Title = "Create";
}
<h2>Create</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <h4>User</h4>
        <hr/>
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.UserName, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.UserName, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.UserName, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.UserPassword, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.UserPassword, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.UserPassword, "", new { @class = "text-danger" })
            </div>
        </div>
        <div class="form-group">
            @Html.LabelFor(model => model.Admin, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                <div class="checkbox">
                    @Html.EditorFor(model => model.Admin)
                    @Html.ValidationMessageFor(model => model.Admin, "", new { @class = "text-danger" })
                </div>
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}
<div>
    @Html.ActionLink("Back to List", "Index")
</div>

```

Рисунок 3.4.2. Вид создания клиента

За вызов данного вида отвечают соответствующие методы на контроллере пользователей(рис. 3.4.3.). Get-метод „Create“, для вызова вида. При нажатие кнопки „Submit“ вызывается post-метод, на который передается модель, свойства которой заполняются из соответствующих полей ввода данного вида.

```

// GET: Users/Create
public ActionResult Create()
{
    return View();
}

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "UserID,UserName,UserPassword,Admin")] User user)
{
    if (ModelState.IsValid)
    {
        db.Users.Add(user);
        db.SaveChanges();
        return RedirectToAction("Index");
    }

    return View(user);
}

```

Рисунок 3.4.3. Методы post-/get Create

Из приведенного выше списка технологий видно, что Microsoft .NET выпускает полновесные архитектуры, для развития как offline так и online приложений. Несмотря на то, что платформа была выпущена в 2002-ом году, как альтернатива платформе Java компании Sun Microsystems (ныне принадлежит Oracle), .NET быстрыми темпами набрал свою популярность среди ит-компаний и их разработчиков. На данный момент (май 2015) язык программирования **C#** занимает 5-ое места в рейтинге сайта www.tiobe.com, однако, его популярность непрерывно растёт.

4. Заключение

В данной работе описан цикл разработки инфостистемы и, соответствующие ей программное обеспечение, с применением Microsoft.NET Framework технологий.

Первая часть дипломной работы описывает проектирование и реализацию десктоп и web приложения для управления пунктом общественного питания. При проектировании ставятся задачи, в расчёт берётся структура данных и соответствующий функционал. На этапе реализации учитываются требования и особенности работы обслуживающего персонала, а также администратора/менеджера. Десктоп приложение является инструментом, предоставляющим все необходимое для создания заказа, а также приёма товара на склад. Web – приложение позволяет следить за складом, а так же создавать новых пользователей.

Также работа позволяет оценить часть инструментария, представленных в .NET Framework(WPF, Entity Framework, ASP.Net MVC), на примере разработанных приложений. Во второй части дипломной работы рассматриваются основные преимущества выше указанных технологий.

В рамках работы не ставилась цель показать все главные особенности Microsoft.NET, но показать наиболее важные для разработчиков бизнес-приложений.

Resümee

Selles lõputöös on kirjeldatud tarkvara väljatöötamise tsükliid, kasutades Microsoft.NET Framework tehnoloogiad.

Bakalaureusetöö esimene osa kirjeldab desktop- ja web-rakenduse projekteerimise ja realiseerimise sõgikohta juhtimise jaoks. Projekteerimise käigus ülesanneks püstitatakse andmete strukturaalne analüüs ja vaadeldakse tuleviku rakenduste funktsionaalsus. Realiseerimise etapil arvestatakse etteantud personali töötajate ja juhataja nõuetega. Desktop rakendus on vahend, mis sisaldab kõike vajalikku, tellimuse loomiseks ja kauba lattu vastu võtmiseks. Web - rakendus mis võimaldab teil jälgida ladu seisundi, samuti luua uusi kasutajaid.

Lõputöö võimaldab hinnata tööriistad esitatud Microsoft.NET Framework (WPF, Entity Framework, ASP.Net MVC), väljatöötatud tehnoloogia rakendamise eeskujul. Töö teises osas vaadeldakse põhilisi Microsoft.NET-i eeliseid ja kasutamise võimalusi.

Lõputöö eesmärgiks ei olnud välja selgitada kõiki põhilisi Microsoft.NET omadusi, kuid tuua välja kõige vajalikumaid eelistusi, mis on uue raamistiku äri-rakenduste arendajate ja kasutajate jaoks olulised.

Summary

This paper describes the development cycle information system and its appropriate software, using Microsoft.NET Framework technology.

The first part of the thesis describes the design and implementation of desktop and web application to manage catering. During designing, put tasks taken into account the structure of the data and meets both functional needs. At the stage of implementation are taken into account the requirements and behavior of the staff as well as administrator / manager. The desktop application is a tool that provides everything needed to create orders and receiving goods at the warehouse. Web - application allows you to monitor the warehouse, as well as to create new users.

Also, to evaluate the work of the tools presented in the .NET Framework (WPF, Entity Framework, ASP.Net MVC), the example of the developed applications. In the second part of the thesis discusses the main advantages of the above technologies.

There was not goal to show all the main features of Microsoft.NET, but the most important were shown for the development of business applications.

Список использованной литературы

1. Charles Petzold “Application = code + markup. A guide to the Microsoft Windows Presentation Foundation”, Microsoft Press; 1 edition (August 16, 2006)
2. Matthew MacDonald “Pro WPF in C# 2010: Windows Presentation Foundation in .NET 4.0”, Apress; 3 edition (March 31, 2010)
3. Gary Hall “Pro WPF and Silverlight MVVM: Effective Application Development with Model-View-ViewModel (Expert's Voice in WPF)”, 2010 edition (December 27, 2010)
4. Rod Stephens “Pro ASP.NET MVC 4” , Apress; 3 edition (March 31, 2010)
5. Глоссарий терминов .NET [WWW]
<http://www.interface.ru/home.asp?artId=2014> (20.11.2010)
6. MSDN MICROSOFT [WWW]
<http://msdn.microsoft.com/ru-ru/library/hfa3fa08.aspx> (20.11.2010)
7. Wikipedia [WWW]
http://ru.wikipedia.org/wiki/C_Sharp (20.11.2010)
8. MSDN MICROSOFT [WWW]
[http://msdn.microsoft.com/ru-ru/library/cc221408\(VS.95\).aspx](http://msdn.microsoft.com/ru-ru/library/cc221408(VS.95).aspx) (20.11.2010)
9. CIT Forum [WWW]
<http://citforum.ru/programming/digest/directxwhatis.shtml> (20.11.2010)
10. Wikipedia [WWW]
<http://en.wikipedia.org/wiki/SQL> (20.11.2010)

11. Wikipedia [WWW]

http://en.wikipedia.org/wiki/Интерфейс_пользователя (20.11.2010)

12. Wikipedia [WWW]

https://en.wikipedia.org/wiki/Model_View_ViewModel (21.05.2015)

13. Wikipedia [WWW]

http://en.wikipedia.org/wiki/ASP.NET_Razor_view_engine (21.05.2015)

14. Wikipedia [WWW]

<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>
(20.11.2015)

15. MSDN MICROSOFT [WWW]

<http://msdn.microsoft.com/ru-ru/library/aa970268.aspx> (20.11.2010)

16. WPFTUTORIAL.NET [WWW]

<http://wpftutorial.net/RoutedEvents.html> (20.11.2010)

17. MSDN MICROSOFT [WWW]

[http://msdn.microsoft.com/en-us/library/ms742806\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms742806(VS.85).aspx) (20.11.2010)

18. DOTNET-GUIDE.COM [WWW]

<http://www.dotnet-guide.com/data-binding-overview.html> (20.11.2010)

Приложение

MenuItemWindow.cs - Код описывающий логику окна составления единицы меню и её редактирования.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using SushiSun.ViewModel;
using SushiSun.Model;
using SushiSun.DAL;

namespace SushiSun
{
    /// <summary>
    /// Interaction logic for MenuItemWindow.xaml
    /// </summary>
    public partial class MenuItemWindow : Window
    {
        #region private fields
        private MenuItemVM _menuItemVM;
        private MenuItemBL _menuItemBL;
        private Product _selectedProduct;
        private ProductMenuItem _selectedProductMenuItem;
        private ProductMenuItem _selectedProductMenuItemEdit;
        private SushiSun.Model.MenuItem _selecteMenuItem;

        #endregion
        #region main
        public MenuItemWindow()
        {
            InitializeComponent();
            _menuItemVM = new MenuItemVM();
            _menuItemBL = new MenuItemBL();
            _menuItemVM.getData();
            this.DataContext = _menuItemVM;
        }

        #endregion
        #region window events
        private void Window_Closing(object sender,
System.ComponentModel.CancelEventArgs e)
        {
            var window = new MainWindow();
            window.Show();
        }
        #endregion
        #region butttons
        private void btnChangeStatus_Click(object sender, RoutedEventArgs e)
```

```

    {
        _selecteMenuItem = lbEditMenuItem.SelectedItem as
        SushiSun.Model.MenuItem;
        if (_selecteMenuItem != null)
        {
            _menuItemVM.MenuItems.SingleOrDefault(x => x.MenuItemId ==
            _selecteMenuItem.MenuItemId).Active
                = _menuItemBL.changeStatus(_selecteMenuItem.MenuItemId,
            _selecteMenuItem.Active);
            lbEditMenuItem.Items.Refresh();
        }
    }

    private void btnMenuItemAdd_Click(object sender, RoutedEventArgs e)
    {
        decimal quantity;
        decimal.TryParse(tbQuantity.Text, out quantity);
        if (_menuItemVM.ProductMenuItems.Count == 0)
        {
            ProductMenuItem productMI = new ProductMenuItem { Product =
            _selectedProduct, Quantity = quantity };
            _menuItemVM.ProductMenuItems.Add(productMI);
            tbQuantity.Clear();
            lbProducts.SelectedIndex = -0;
            lbMenuItems.Items.Refresh();
            isAddEnable();
        }
        else
        {
            bool found = false;
            foreach (ProductMenuItem mi in _menuItemVM.ProductMenuItems)
            {
                if (mi.Product.ProductId == _selectedProduct.ProductId)
                {
                    mi.Quantity = mi.Quantity + quantity;
                    tbQuantity.Clear();
                    lbProducts.SelectedIndex = -0;
                    isAddEnable();
                    found = true;
                    lbMenuItems.Items.Refresh();
                    break;
                }
            }
            if (!found)
            {
                ProductMenuItem productMI = new ProductMenuItem { Product =
            _selectedProduct, Quantity = quantity };
                _menuItemVM.ProductMenuItems.Add(productMI);
                tbQuantity.Clear();
                lbProducts.SelectedIndex = -0;
                isAddEnable();

                lbMenuItems.Items.Refresh();
            }
        }
        isSubmitEnable();
    }

    private void btnMenuItemRemove_Click(object sender, RoutedEventArgs e)
    {

```

```

        _selectedProductMenuItem = lbMenuItems.SelectedItem as
ProductMenuItem;
        if (_selectedProductMenuItem != null)
        {
            _menuItemVM.ProductMenuItems.Remove(_selectedProductMenuItem);
            lbMenuItems.SelectedIndex = -1;
            lbMenuItems.Items.Refresh();
            isAddEnable();
            isSubmitEnable();
        }
    }

    private void btnSubmit_Click(object sender, RoutedEventArgs e)
    {
        Decimal p;
        Decimal.TryParse(tbMenuItemPrice.Text, out p);
        string name = tbMenuItemName.Text;

        _menuItemVM.MenuItems.Add(_menuItemBL.initializeMenuItem(_menuItemVM.ProductMenuI
tems, name, p));
        _menuItemVM.ProductMenuItems.Clear();
        lbEditMenuItem.Items.Refresh();
        tbMenuItemName.Clear();
        tbMenuItemPrice.Clear();
        tbQuantity.Clear();
    }

    private void btnAddProduct_Click(object sender, RoutedEventArgs e)
    {
        decimal quantity = 0.1m;
        ProductMenuItem toAdd = new ProductMenuItem()
        {
            ProductId = _selectedProduct.ProductId,
            MenuItemId = _selecteMenuItem.MenuItemId,
            Product = _selectedProduct,
            Quantity = quantity
        };

        _menuItemVM.ProductMenuItemsEdit.Add(toAdd);
        lbEditProductMenuItem.SelectedIndex = -1;
        cbAddProduct.SelectedIndex = -1;
        lbEditProductMenuItem.Items.Refresh();
        _selectedProduct = null;
        editAddEnable();
        btnSave.IsEnabled = true;
    }

    private void btnSave_Click(object sender, RoutedEventArgs e)
    {
        var newItems =
        _menuItemBL.changeItem2(_menuItemVM.ProductMenuItemsEdit.ToArray(),
        _selecteMenuItem.MenuItemId);
        _menuItemVM.MenuItems.SingleOrDefault(x => x.MenuItemId ==
        _selecteMenuItem.MenuItemId).ProductMenuItems.Clear();

        foreach (var item in newItems)
        {
            _menuItemVM.MenuItems.SingleOrDefault(x => x.MenuItemId ==
        _selecteMenuItem.MenuItemId).ProductMenuItems.Add(item);
        }
    }

```

```

        lbEditProductMenuItem.SelectedIndex = -1;

        lbEditMenuItem.SelectedIndex = -1;

        lbEditProductMenuItem.Items.Refresh();
        lbEditMenuItem.Items.Refresh();
        btnSave.IsEnabled = false;
    }

    private void btnSetNewQuantity_Click(object sender, RoutedEventArgs e)
    {
        decimal q;
        decimal.TryParse(tbEditQuantity.Text, out q);
        _menuItemVM.ProductMenuItemsEdit.SingleOrDefault(x => x.ProductId ==
_selectedProductMenuItemEdit.ProductId).Quantity = q;
        lbEditProductMenuItem.SelectedIndex = -1;
        lbEditProductMenuItem.Items.Refresh();
        btnSave.IsEnabled = true;
    }

    private void btnRemoveProduct_Click(object sender, RoutedEventArgs e)
    {
        _menuItemVM.ProductMenuItemsEdit.Remove(_selectedProductMenuItemEdit);
        lbEditProductMenuItem.SelectedIndex = -1;
        lbEditProductMenuItem.Items.Refresh();
        btnSave.IsEnabled = true;
    }
    #endregion
    #region buttons validation
    private void lbProducts_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
    {
        _selectedProduct = lbProducts.SelectedItem as Product;
        isAddEnable();
    }

    private void tbQuantity_TextChanged(object sender, TextChangedEventArgs
e)
    {
        isAddEnable();
    }

    private void tbMenuItemName_TextChanged(object sender,
    TextChangedEventArgs e)
    {
        isSubmitEnable();
    }

    private void tbMenuItemPrice_TextChanged(object sender,
    TextChangedEventArgs e)
    {
        isSubmitEnable();
    }

    private void lbMenuItems_SelectionChanged(object sender,
    SelectionChangedEventArgs e)
    {
        isSubmitEnable();
    }

```

```

        private void lbEditMenuItem_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            _menuItemVM.ProductMenuItemsEdit.Clear();
            _selecteMenuItem = lbEditMenuItem.SelectedItem as
SushiSun.Model.MenuItem;
            if (_selecteMenuItem != null)
            {
                List<ProductMenuItem> productMenuItemsEdit =
_selecteMenuItem.ProductMenuItems.ToList();
                foreach (var productMenuItem in productMenuItemsEdit)
                {
                    _menuItemVM.ProductMenuItemsEdit.Add(productMenuItem);
                }
                lbEditProductMenuItem.Items.Refresh();
            }
            btnSave.IsEnabled = false;
        }

        private void lbEditProductMenuItem_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            _selectedProductMenuItemEdit = lbEditProductMenuItem.SelectedItem as
ProductMenuItem;
            isEditingEnabale();
        }

        private void tbEditQuantity_TextChanged(object sender,
TextChangedEventArgs e)
        {
            isSetEnable();
        }

        private void ComboBox_SelectionChanged(object sender,
SelectionChangedEventArgs e)
        {
            _selectedProduct = cbAddProduct.SelectedItem as Product;
            editAddEnable();
        }

        private void isAddEnable()
        {
            Decimal q;
            if (_selectedProduct != null && Decimal.TryParse(tbQuantity.Text, out
q) && q > 0)
            {
                btnMenuItemAdd.IsEnabled = true;
            }
            else { btnMenuItemAdd.IsEnabled = false; }
        }

        private void isSubmitEnable()
        {
            Decimal p;
            if (_menuItemVM.ProductMenuItems.Count != 0 &&
Decimal.TryParse(tbMenuItemPrice.Text, out p) &&
!string.IsNullOrEmpty(tbMenuItemName.Text))
            {
                btnSubmit.IsEnabled = true;
            }
            else { btnSubmit.IsEnabled = false; }
        }

```

```

    }

    private void isEditingEnabale()
    {
        if (_selectedProductMenuItemEdit != null)
        {
            tbEditQuantity.IsEnabled = true;
            btnRemoveProduct.IsEnabled = true;
            tbEditQuantity.Text =
                _selectedProductMenuItemEdit.Quantity.ToString();
        }
        else
        {
            tbEditQuantity.IsEnabled = false;
            btnRemoveProduct.IsEnabled = false;
            tbEditQuantity.Clear();
        }
    }

    private void editAddEnable()
    {
        if (_selectedProduct != null)
        {
            bool found = false;
            foreach (var produvtMenuItem in
                _selecteMenuItem.ProductMenuItems)
            {
                if (produvtMenuItem.ProductId == _selectedProduct.ProductId)
                {
                    found = true;
                    break;
                }
            }
            if (found == true)
            {
                btnAddProduct.IsEnabled = false;
            }
            else
            {
                btnAddProduct.IsEnabled = true;
            }
        }
    }

    private void isSetEnable()
    {
        decimal q;
        if (_selectedProductMenuItemEdit != null &&
            Decimal.TryParse(tbEditQuantity.Text, out q) && q > 0 && q !=
                _selectedProductMenuItemEdit.Quantity)
        {
            btnSetNewQuantity.IsEnabled = true;
        }
        else
        {
            btnSetNewQuantity.IsEnabled = false;
        }
    }
}
#endregion
}
}

```

