|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Приднестровский государственный университет им. Т.Г. Шевченко | | | | | | | | | |
| Инженерно-технический институт | | | | | | | | | |
| Инженерно-технический факультет | | | | | | | | | |
| Кафедра Программное обеспечение вычислительной техники  и автоматизированных систем | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **КУРСОВАЯ РАБОТА** | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |
| по дисциплине «Сетевые технологии» | | | | | | | | | |
|  | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |
|  | | | | | | | | | |
| **тема: «Сетевое приложение учета работы предприятия по изготовлению мягкой мебели»** | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | Работу выполнил | | | |
|  |  |  |  |  |  | студент группы ИТ13ДР62ПИ1 | | | |
|  |  |  |  |  |  | Федоров Артем Андреевич | | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | Руководитель, | | | |
|  |  |  |  |  |  | доцент, к.п.н. | | | |
|  |  |  |  |  |  | Нижегородова Маргарита Владимировна | | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | | | | | | | | | |

Тирасполь, 2016

**ТЕКСТ ИСХОДНОГО ЗАДАНИЯ**

**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
| ВВЕДЕНИЕ | | 4 |
| 1 ИССЛЕДОВАНИЕ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ | | 5 |
|  | 1.1 Описание поставленной задачи, ее обоснование | 5 |
|  | 1.2 Актуальность исследуемой задачи и обзор методов решения подобных задач | 5 |
|  | 1.3 Современное состояние исследуемой задачи | 6 |
|  | 1.4 Постановка задачи, системные требования, требования к входным данным и выходным формам | 6 |
| 2 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ И АРХИТЕКТУРЫ  ПРОГРАММНОГО ПРОДУКТА | | 9 |
|  | 2.1 Выбор методов и средств для реализации | 9 |
|  | 2.2 Описание применяемых алгоритмов | 10 |
|  | 2.3 Структура, архитектура программного продукта | 10 |
|  | 2.4 Описание логической структуры программного продукта | 10 |
|  | 2.5 Функциональная схема, функциональное назначение программного продукта | 11 |
| 3 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА | | 12 |
|  | 3.1 Описание реализации | 12 |
|  | 3.2 Описание пользовательского интерфейса | 16 |
|  | 3.4 Полученные результаты и их анализ | 21 |
| ЗАКЛЮЧЕНИЕ | | 22 |
| СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ | | 23 |
| ПРИЛОЖЕНИЕ А – Листинг программы | | 24 |

**ВВЕДЕНИЕ**

**1 ИССЛЕДОВАНИЕ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

**1.1 Описание поставленной задачи, ее обоснование**

Целью данной курсовой работы является создание сетевой информационной системы.

Для проектируемой предметной области необходимо:

1. Разработать клиент-серверное приложение, где серверной частью будет являться база данных, реализованная средствами СУБД. Клиентское приложение должно быть реализовано с помощью *Visual Studio 2005*.

2. База данных должна содержать информацию об продаваемом газовом оборудовании и предоставляемом сервисе, сотрудниках фирмы и об ее клиентах, а также о поступающих заказах.

3. Информационная система должна содержать второе клиентское приложение, представляющее собой веб-приложение, обращающееся к серверному приложению по протоколу *HTTP* и реализованное с помощью технологии *ASP.NET*.

Сетевое приложение должно выполнять следующие функции:

1. На стороне сервера:

- сохранение в файл статистики подключений клиентских приложений;

- обработка запросов к базе данных (далее БД), осуществление расчетов с помощью входных параметров из клиентских приложений;

- удаление данных доступно только со стороны серверного приложения.

2. На стороне клиентского *Windows*-приложения:

- запись в базу данных новых данных о сотрудниках фирмы, имеющейся в наличии продукции и сервисе, редактирование данных в таблицах, удаление данных не разрешается;

- поиск имеющихся данных по не менее, чем двум критериям;

- приложение содержит разнообразные элементы управления и подсказки пользователю.

3. На стороне веб-приложения:

- поиск информации об имеющейся продукции по не менее чем двум критериям;

- возможность добавления новых заказов в БД, а также редактирование информации о данных зарегистрированных пользователей;

- проверка правильности ввода данных в веб-форму с помощью элементов валидаторов.

Так же в информационной системе должна присутствовать аутентификация и авторизация как сотрудников предприятия, так и его клиентов.

**1.2 Обоснование актуальности исследуемой задачи и обзор методов решения подобных задач**

Проектируемая предметная область является актуальной, в связи со стремительным развитием информационных технологий.

Клиенту фирмы должно быть удобно сделать заказ на газовое оборудование и его сервис, а сотрудникам фирмы обработать входящие заказы от клиентов. Это удобнее всего сделать с использованием сетевого приложения.

Задание к данной курсовой работе подразумевает использование *.NET Framework* для реализации приложения.

Но существуют и другие средства осуществления подобных систем. При реализации можно использовать в качестве основного языка программирования не только *C#,* но и *Python, PHP, Ruby* и другие языки программирования. При этом используются различные технологии, Фреймворки и каркасы для создания программного обеспечения.

Например, *Django –* свободный программный каркас для веб-приложений на языке *Python. WordPress –* система управления содержимым сайта с открытым исходным кодом, написанная на *PHP*[1].

Так же при реализации для отладки и тестирования приложений и их взаимодействия в данной курсовой работе используется *IIS (Internet Information Server)* от компании Майкрософт. Альтернативой ему можно считать *Apache HTTP*-сервер. Его преимуществами являются кроссплатформенность и гибкость конфигурации.

**1.3 Современное состояние исследуемой задачи**

Многие фирмы, предоставляющие ремонт и продажу газового оборудования размещают каталоги и информацию о своих услугах в сети Интернет. Это значительно ускоряет и упрощает работу с клиентами.

На текущий момент существует огромное количество веб-ресурсов по исследуемой задаче. Существуют веб-сайты различной сложности. Например, сайт «*vdgu.ru*» реализован с помощью *html*-страниц без использования серверного программирования. На рисунке 1.1 представлена страница данного сайта, содержащая информацию о мягкой мебели.



Рисунок 1.1 – Страница сайта *vdgu.ru*

Можно отметить следующие особенности данного сайта: вся информации об имеющейся продукции вбивается вручную непосредственно в код, на сайте отсутствует регистрация, дизайн обычный.

ЕЩЕ 1 ПРИМЕР

**1.4 Постановка задачи, системные требования, требования к входным данным и выходным формам**

Предметная область данной курсовой работы подразумевает разработку сетевой информационной системы, состоящей из трех подсистем:

1) серверное приложение;

2) клиентское приложение;

3) веб-приложение.

Серверное приложение будет представлять собой веб-сервис, взаимодействующий с БД *SQL,* и обрабатывающий входящие запросы от клиентских приложений.

Первое клиентское приложение будет реализовано с помощью *Windows Forms* и представлять собой интерфейс для сотрудников фирмы. Его функциями будет добавление и изменение информации о сотрудниках, клиентских заказах, услугах фирмы, имеющейся продукции.

Входными формами в данном приложении будут формы с информацией о сотрудниках, товарах, заказах и материалах предприятия, а выходные формы – отчеты о поступлении новых заказов и о выполнении и доставке заказов клиентам.

Второе клиентское приложение будет реализовано с помощью *ASP.NET* и представлять собой полноценный сайт, ориентированный на клиентов фирмы. Это приложение предоставит клиентам-пользователям сайта следующую функциональность:

- просмотр информации об уже имеющейся продукции и о предоставляемых услугах;

- добавление продукции и услуг в корзину товаров;

- возможность осуществления заказов.

Входными формами этого приложения будут поля регистрации и авторизации пользователя, поля для поиска продукции с помощью различных фильтров, а выходными формами – отчеты ???????????????????????????.

Для использования данной информационной системы были выявлены следующие необходимые системные требования:

- операционная система *Windows* *7, 8, 10*;

- *2Gb* ОЗУ;

- любой веб-браузер;

- *100Mb* дополнительного места на жестком диске.

Дополнительные требования:

- разрешение экрана не менее, чем 1280x720 пикселей;

- видеоадаптер *DirectX 9* или более поздней версии.

**2 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ И АРХИТЕКТУРЫ**

**ПРОГРАММНОГО ПРОДУКТА**

**2.1 Выбор методов и средств для реализации**

Данная система будет реализована с помощью *Visual Studio 2015*, использованием *.NET Framework* версии 4.5.2. Эта среда выбрана в связи с наилучшими средствами для разработки сетевых приложений.

Для реализации серверного приложения было принято использовать веб-сервис. Альтернативой выбранному методу является использование *.NET Remoting.* Решение использовать веб-сервисы обусловлено тем, что приложение с таким подходом в перспективе имеет возможность взаимодействия с другими платформами и операционными системами.

В качестве базы данных для проекта решено использовать *SQL Server*, так как это средство разработки имеет хорошую производительность и интеграцию в *Visual Studio*.

Клиентское приложение для сотрудников было принято разрабатывать с помощью *Windows Forms*, так как этих средств разработки более чем достаточно для рабочего приложения, используемого сотрудниками предприятия.

Для разработки веб-приложения для клиентов предприятия было решено использовать *ASP.NET 5*, потому что данное приложение является последней и лучшей версией для разработки на основе *Web Forms.*

Так же при реализации было решено использовать в качестве дополнительного языка программирования *JavaScript* с использованием библиотек *JQuery*, *JQuery UI* и подходом к построению интерактивных пользовательских интерфейсов веб-приложений *AJAX.* Данные средства разработки помогут улучшить внешний вид и удобство пользования клиентским приложением.

Оба клиентских приложения взаимодействуют с серверным приложением посредством протокола *HTTP* с помощью спецификации *SOAP,* используя *XML* документы.

**2.2 Описание применяемых алгоритмов**

Работа обоих клиентских приложений с базой данных происходит следующим образом: клиентские приложения, обращаются к серверному, вызывая методы веб-службы с какими-либо параметрами. Например, при работе с продукцией предприятия это может быть наименование товара и его цена, а при работе с заказами – имя заказчика, его телефон, адрес и т.д. Методы веб-службы, получая на вход значения, применяют их в вызванном методе, подставляя в запрос либо выполняя с ними еще какие-либо действия. После выполнения всех действий метод веб-службы в большинстве случаев возвращает какую-либо информацию на сервер. Это может быть таблица данных, либо логическое значение *true/false,* либо просто некоторая строка. Клиентское приложение получает данную информацию и вносит соответствующие изменения согласно коду приложения.

Реализация поиска в клиентских приложениях происходит следующим образом: пользователь вводит информацию в поле и выбирает диапазон цен. Нажатие на кнопку поиска инициирует запрос на сервер, содержащий выбранные пользователем данные, который генерирует на основе полученных данных *SQL* запрос и присылает выборку из базы данных. Изменение и добавление данных в базу реализовано аналогично поиску, с тем лишь отличием, что пользователь не получает в ответ выборку данных, а лишь получает ответ об успешном или неуспешном изменении, добавлении данных.

В веб-приложении у пользователя есть возможность выбрать товары и их количество с помощью корзины. Заходя на веб-сайт для каждого клиента организуется своя клиентская сессия, содержащая пары ключ-значение. Нажимая на кнопку добавить в корзину под соответствующей продукцией, клиент инициирует добавление товара в коллекцию, хранящуюся в сессии под ключом «*shop*». Заходя на страницу с корзиной, пользователь видит отображение всех выбранных товаров и итоговую цену, которая складывается в зависимости от цены и количества каждого добавленного в корзину товара. Также, с помощью нажатия на кнопку доступна функция удаления товаров из корзины.

**2.3 Структура и архитектура программного продукта**

В ходе проектирования программного продукта была выявлена общая архитектурная схема программного продукта, представленная на рисунке 2.1.

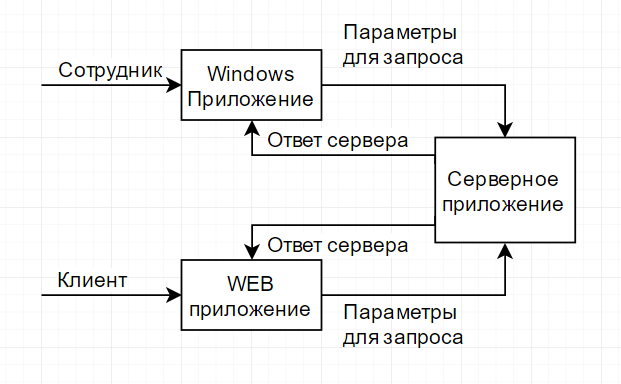


Рисунок 2.1 – Архитектурная схема программного продукта

По данному рисунку видно, что архитектура сетевого приложения включает в себя 3 подсистемы: клиентское веб-приложение, *Windows*-приложение для сотрудников и серверное приложение.

Как правило, клиентские приложения вызывают с сервера необходимый метод с требуемыми им параметрами. Сервер же, получив такой вызов, выполняет данный метод, и в зависимости от типа возвращаемого значения, отправляет данные обратно клиентскому приложению.

На рисунке 2.2 представлена структурная схема веб-приложения.

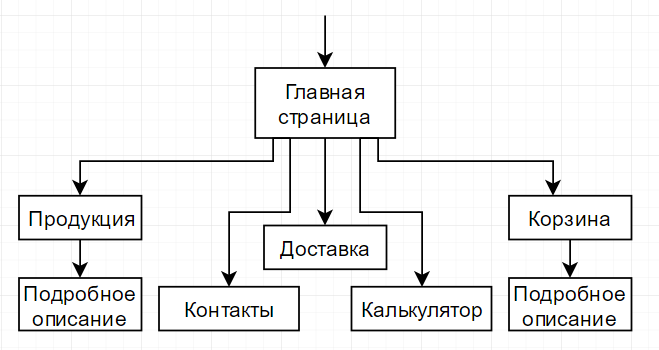


Рисунок 2.2 – Структура веб-приложения

Веб-приложение состоит из страниц формата *aspx.* Каждая страница шаблонизирована с помощью мастер страницы *MasterPage.Master.* Мастер страница позволяет не повторять однотипные структуры в каждой страницы отдельно.

«Главная страница» включает общую информацию о предприятии. Страница продукции содержит каталог предприятия. «Подробное описание» продукции позволяет пользователю увидеть полное описание выбранного из каталога товара. С данных страниц пользователь имеет возможность добавить понравившиеся товары в корзину, после чего посмотреть их отображение на странице с аналогичным названием «Корзина»; также на данной странице можно осуществить заказ выбранных товаров.

Страница «Контакты» и «Доставка» содержат справочную информацию о местонахождении предприятия и склада, а также стоимости дополнительных услуг.

Страница «Калькулятор» предоставляет возможность клиенту предприятия

На рисунке 2.3 представлена структура *Windows*-приложения.

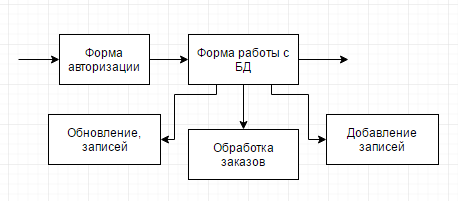


Рисунок 2.3 – структурная схема *Windows*-приложения

*Windows*-приложение реализовано с помощью форм *Windows.* Данная подсистема содержит:

1) форму авторизации, так как приложение доступно только сотрудникам предприятия;

2) основную форму – форму работы с базой данных, на данной форме доступен поиск по таблицам из БД;

3) форму для добавления информации в таблицы базы данных;

4) форму для обновления информации в базе данных;

5) форму для работы с заказами.

Формы для добавления и обновления информации генерируются автоматически в зависимости от выбранной таблицы и строки в ней.

**2.4 Описание логической структуры программного продукта**

Взаимодействие веб-сервиса и клиентских приложений является основой логической структуры проекта данной курсовой работы.

Используя веб-сервисы с помощью *Visual Studio*, появляется возможность связывания приложений с помощью протокола доступа к объектам *SOAP,* используя протокол прикладного уровня передачи гипертекстовых сообщений *HTTP.* Для передачи сообщений с помощью этого протокола используется язык *XML.* Файл на языке *XML* представляет собой набор метаданных, необходимых для корректной передачи сообщения и самой передаваемой информации.

Формирование *XML* документов вручную является сложным и бессмысленным процессом, так как *Visual Studio* позволяет формировать их автоматически. Для этого необходимо сериализуемый метод пометить как *WebMethod.*

При создании веб-сервиса был реализован класс *MebelService*. Этот класс представляет собой группу веб-методов, взаимодействующих с клиентскими приложениями, а также методы, использующиеся для локального вызова. На рисунке 2.4 представлена диаграмма класса *MebelService.*



Рисунок 2.4 – Диаграмма класса *MebelService*

Как видно на диаграмме, в данном классе реализован лишь один локальный метод, отвечающий за разбиение входящих параметров на группы – имя параметра, значение параметра.

При реализации клиентского веб-приложения были разработаны классы *MebelSearching* и *ShopCart*. Диаграммы классов представлены на рисунке 2.6.

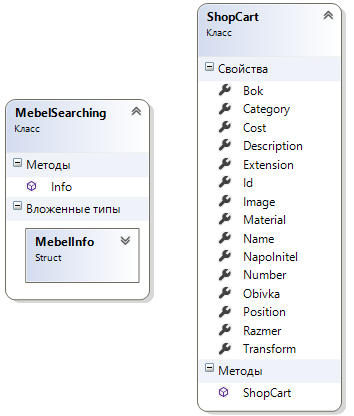


Рисунок 2.6 – Диаграммы классов *MebelSearching* и *ShopCart*

Класс *MebelSearching* отвечает за приведение данных, полученных с веб-сервиса. Класс содержит структуру данных *MebelInfo*, представляющую набор колонок таблицы *Mebel* и метода *Info*, типа *List<MebelInfo>* со входным параметром типа *DataTable.* Метод получает на вход выборку из таблицы *Mebel* и возвращает ее же преобразованную в виде листа.

Второй класс *ShopCart* предназначен для работы с продуктовой корзиной пользователя клиентского приложения. Класс содержит свойства, необходимые для отображения товаров, содержащихся в корзине, а также конструктор, с помощью которого можно создать такой товар.

В *Windows*-приложении были реализованы четыре класса: *ColumnName, ColumnSet, ColumnTableSet, Validation.* Диаграмма этих классов представлена на рисунке 2.7.

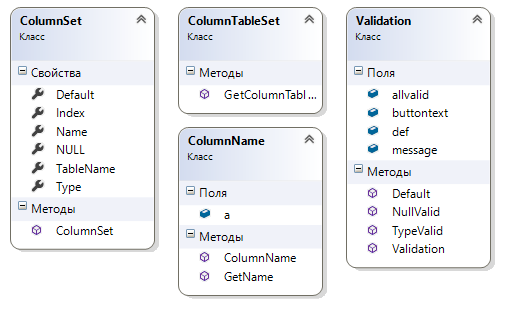


Рисунок 2.7 – Диаграммы классов

Класс *ColumnSet* предназначен для подробного описания колонок базы данных. Класс содержит свойства для имени таблицы, имени колонки, индексе этой колонки в БД, ее типе, значении по умолчанию и допустимости нулевого значения.

Класс *ColumnTableSet* позволяет реализовать таблицу с описанием всех колонок конкретной таблицы БД. Он содержит один метод *GetColumnTableSet,* возвращаемый тип которого *List<ColumnSet>*, входными параметрами являются имя таблицы и количество колонок данной таблицы.

Класс *ColumnName* содержит словарь всех названий колонок БД и более понятных им сопоставлений на русском языке.

С помощью класса *Validation* осуществляется проверка правильности введенных данных в *Windows*-приложении. Методам и конструктору данного класса на вход поступает информация о колонке типа *ColumnSet*, а также значение, которое было введено пользователем. Выходными данными являются значения типа *bool* или сообщение о корректности данных.

**2.5 Функциональная схема, функциональное назначение программного продукта**

Основным назначением приложения является осуществление выбора продукции (мягкой мебели) клиентом и оформление заказа, на сайте, после чего подтверждение и обработка заказа сотрудниками предприятия.

На рисунке 2.5 представлена диаграмма вариантов использования веб-приложения клиентом предприятия.



Рисунок 2.5 – Диаграмма вариантов использования веб-приложения

По данной диаграмме видно, что у клиента есть возможность посетить главную страницу и страницу контактов для получения информации о предприятии. Так же имеется возможность посчитать примерную стоимость мебели.

Но для реализации главной функции клиент заходит на страницу с продукцией. На этой странице он осуществляет поиск необходимого товара. Выбранный товар можно добавить в корзину, после чего есть возможность перейти к корзине. Для осуществления заказа клиенту желательно также просмотреть условия доставки, после чего со страницы корзины клиент оформляет заказ.

После оформления заказа в *Windows*-приложение приходит оповещение о появлении нового заказа. После чего осуществляется звонок клиенту, по указанному им номеру, уточнение деталей заказа, его подтверждение и передача на исполнение. После исполнения заказа он будет передан на ожидание доставки. После осуществления передачи продукции клиенту, заказ помечается как завершенный.

**3 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО**

**ПРОДУКТА**

**3.1 Описание реализации**

Выделим наиважнейшие функциональные части приложения.

Для реализации поиска продукции используется кнопка *SearchButton\_Click*. Код данной кнопки представлен в листинге 3.1.

string category = Master.CategoryText;

string name = Master.NameText;

string extension = Master.ExtensionText;

string costFrom = Master.CostFrom;

string costTo = Master.CostTo;

datatable = service.Search(category,name, extension,costFrom,costTo);

BindData();

Листинг 3.1

Параметры поиска считываются из свойств, определенных в мастер странице, после чего происходит вызов серверного метода *Search*, результаты которого помещаются переменную *datatable* типа *DataTable*.

Запрос, формируемый веб-методов Search представлен в листинге 3.2.

if (Category != "")

Category = " where Category like N'%" + Category + "%' ";

else

Category = " where Category like N'%' ";

if (Name != "")

Name = "and Name like N'%" + Name + "%' ";

else Name = "and name like N'%' ";

string Cost = "";

try

{

Cost = "and Cost between " + Convert.ToDouble(from).ToString() + " and " + Convert.ToDouble(to).ToString() + " ";

}

catch { }

if (Extension != "")

Extension = "and Extension like N'%" + Extension + "%'";

string request = "SELECT \* From Mebel" + Category + Name + Cost + Extension;

Листинг 3.2

Для связывания элементов управления с источниками данных в веб-приложении используется метод *BindData*, представленный в листинге 3.3.

MebelView1.DataSource = MebelSearching.Info(datatable);

MebelView1.DataBind();

Листинг 3.3

Примерная стоимость мебели рассчитывается с помощью следующего кода на языке *javascript* с применением библиотеки *jquery,* представленного в листинге 3.4.

$('input.itog').click(function () {

var length = ($('input.length').val()) / 1000;

var width = ($('input.width').val()) / 1000;

var height = ($(''input.height').val()) / 1000;

var kub = length \* width \* height;

var material = $('.material :selected').val(),

obivka = $('.obivka :selected').val(),

napolnitel = $('.napolnitel :selected').val(),

transform = $('.transform :selected').val(),

bok = $('.bok :selected').val();

var itog = kub \* material + kub \* obivka + kub \* napolnitel + transform \* kub + bok \* kub;

$('.cost').text(itog.toFixed(0) + "₽") });

Листинг 3.4

Цена на материалы в базе данных указываются за один квадратный метр, поэтому введенные пользователем данные в указанном выше скрипте делятся на 1000. После этого высчитывается размер товара в кубических метрах. Стоимость же высчитывается исходя из полей, выбранных в соответствующих комбинированных списках (тег *select*). Полученное значение округляется до целого.

Комбинированные списки, содержащие информацию о типе и стоимости каркаса, обивке наполнителе и других параметрах заполняются с помощью *ajax* библиотеки *jquery.* На листинге 3.5 представлен код для заполнения комбинированного списка видами обивки.

function obivkaSelect() {

$.ajax({

type: "POST",

url: this.location.href + "/obivkaSelect",

data: '{}',

contentType: "application/json; charset=utf-8",

dataType: "json",

success: obivka,

}); }

Листинг 3.5

Данные для заполнения берутся в виде *XML* файла из веб-метода в коде страницы. Код веб-метода calculatorSelect, возвращающего данные для калькулятора, представлен в листинге 3.6.

[WebMethod]

public DataSet calculatorSelect(string category)

{

string request = "SELECT Description,Cost FROM Calculator where Category like N'%" + category + "%'";

SqlCommand command = new SqlCommand(request, con);

SqlDataAdapter read = new SqlDataAdapter(command);

DataSet ds = new DataSet();

read.Fill(ds);

return ds;

}

Листинг 3.6

При удачном извлечении данных вызывается метод, указанный после ключевого слова *success*. Для типов обивки этоn метод называется *obivka*. Данный метод ищет в *XML* файле все пары значений, собранных в тегах *Description* (тип обивки) и *Cost* (ее стоимость) и вставляет их в качестве видимого текста и значения в тег *option* внутрь соответствующего тега *select*.

function obivka(response) {

var xmlDoc = $.parseXML(response.d);

var xml = $(xmlDoc);

var type = xml.find("Description");

var cost = xml.find("Cost");

var that = $(".obivka"); var i = 0;

$(mebel).each(function () {

$('<option />', { text: $(this).text(),

value: $(cost[i]).text() }).appendTo(that);

i++; }); i = 0; }

Листинг 3.7

Добавление товаров в корзину происходит при перезагрузке элемента *ListView*. Нажатие на кнопку отслеживается в методе *Page\_Load*.

В начале считывается текущее состояние корзины из сессии с ключом *“shop”* в переменную *shop* типа *List<ShopCart>*, и выявляется позиция (переменная *Position* типа *int*), которая будет присвоена новому элементу. Код представлен в листинге 3.8.

int Position = 1;

if (Session["shop"] as List<ShopCart> != null)

{ shop = (List<ShopCart>)Session["shop"];

Position = shop[shop.Count - 1].Position + 1; }

Листинг 3.8

После из базы данных извлекается таблица со строкой, содержащую добавляемый в корзину элемент.

DataTable addedElement = service.searchById (Convert.ToInt32(Request.Form.Get("add").ToString()), "Mebel");

Листинг 3.9

Из данной строки байтовый массив приводится к виду, который сможет считать браузер.

string Image = addedElement.Rows[0]["Image"] != System.DBNull.Value ?Convert.ToBase64String ((byte[]) addedElement.Rows[0]["Image"], 0, ((byte[]) addedElement.Rows[0]["Image"] as byte[]).Length) :

string.Empty;

Image = "data:image/jpeg;base64," + Image;

Листинг 3.10

После этого реализуется непосредственно создание элемента типа *ShopCart* и его добавление к существующим и записи существующих в сессию. Код представлен в листинге 3.11

ShopCart add = new ShopCart(Position, Convert.ToInt32(addedElement.Rows[0]["Id"]),

Convert.ToDouble(addedElement.Rows[0]["Cost"].ToString()), 1,

addedElement.Rows[0]["Category"].ToString(), addedElement.Rows[0]["Name"].ToString(),

Продолжение листинга 3.11

addedElement.Rows[0]["Extension"].ToString(), addedElement.Rows[0]["Description"].ToString(),

Image, addedElement.Rows[0]["Razmer"].ToString(),

addedElement.Rows[0]["Obivka"].ToString(), addedElement.Rows[0]["Napolnitel"].ToString(),

addedElement.Rows[0]["Transform"].ToString(), addedElement.Rows[0]["Bok"].ToString(),

addedElement.Rows[0]["Material"].ToString());

shop.Add(add);

Session["shop"] = shop;

Листинг 3.11

Удаление из корзины происходит с помощью метода *removeatshop*, представленного в листинге 3.12.

shop = new List<ShopCart>();

if (Session["shop"] as List<ShopCart> != null)

shop = (List<ShopCart>)Session["shop"];

for (int i = 0; i < shop.Count; i++)

{

if (shop[i].Id == Convert.ToInt32 (Request.Form.Get("removefcart").ToString()))

{ shop.RemoveAt(i); break; } }

Session["shop"] = shop;

BindData();

Листинг 3.12

Считываются из сессии *Session[“shop”]* данные о находящихся в корзине товарах, после этого в цикле происходит поиск элемента, с которым связана нажатая пользователем кнопка *removefromcart* и его удаление с помощью встроенного метода *RemoveAt*.

Функция расчета итоговой стоимости происходит с помощью *javascript* функции *itogo,* представленной в листинге 3.12.

$('.label1.foot1').text(function () {

var all = 0;

var cost = $('.costText');

var number = $('.numberText');

for (var i = 0; i < cost.length; i++) {

var cos = parseInt(($(cost[i]).text()).replace(/\D+/g, ""));

all += cos \* $(number[i]).val();}

return "Итого: " + all + " ₽"; });

Листинг 3.12

В переменные *cost* и *number* записываются все данные о стоимостях и количествах во всех строках элемента *GridView*. Далее для каждой строки в итоговую стоимость добавляется произведение стоимости товара на его количество.

Проверка правильности данных, введенных пользователем *Windows*-приложения, происходит с помощью класса *Validation*.

Данный класс содержит три поля, указанных в листинге 3.13. Поле *message* типа *string* служит для вывода сообщения о правильности введенных пользователем данных. Поле *allvalid* служит для возвращения логического значения, правильные введены данные или нет. Поле *def* служит для того, чтобы узнать, содержит ли данное поле значение по умолчанию.

public static bool def = false;

public bool allvalid = false;

public string message = "";

Листинг 3.13

В классе Validation реализовано три статических метода.

Метод *TypeValid,* представленный в листингше 3.14 принимает два аргумента, первый аргумент это введенные пользователем данные типа *string*, а второй – информация о типе колонке, в которую вводятся данные, тип этого поля *ColumnSet*.

switch (b.Type)

{ case "int":try { int.Parse(a); }

catch { return false; }

return true;

case "money":try { float.Parse(a); }

catch { return false; }

return true;

case "bit":a = a.ToLower();

if (a == "0" || a == "1"||a=="true"||a=="false")

return true;

else return false;

case "date":try { DateTime.Parse(a); }

catch { return false; }

return true; }

return true;

Листинг 3.14

Метод смотрит на тип введенных данных с помощью оператора *switch* и проверяет возможность их приведения к данному типу.

Метод *NullValid,* указанный в листинге 3.15 проверяет правильность введенных пользователем данных с точки зрения нулевого значения.

if (b.NULL && a == "")return false;

else return true;

Листинг 3.15

Метод *DefaultValid* позволяет определить, есть ли у данного поля значение по умолчанию.

int l = 0;

try

{ l = b.Default.Length; }

catch

{ }

if (l != 0)

return false;

else

{ return true; }

Листинг 3.16

Так же рассматриваемый класс Validation содержит конструктор.

if (!NullValid(a, b))

{

message = "должна иметь ненулевое значение!";

if (DefaultValid(b))

{ def = true; }

}

else

if (!TypeValid(a, b))

message = "имеет неправильный тип";

else

{

message = "прошла проверку";

allvalid = true;

}

Листинг 3.17

В конструкторе вызываются все методы класса и в зависимости от возвращаемых ими значений, принимают свои значения поля класса.

3.2 Описание пользовательского интерфейса

Открывая клиентское веб-приложение, пользователь видит главную страницу сайта *main.aspx*. Скриншот главной страницы показан на рисунке 3.1.



Рисунок 3.1 – Главная страница сайта

На главной странице расположена информация о компании и ассортименте товаров, представленных на сайте.

В верхней части страницы представлены логотип компании и меню сайта. *Html* код данной части представлен в листинге 3.18.

<header>

<div class="logo">

<div class="centerall">

<div class="imglogo"><img src="../img/logo.png"/></div>

<div class="notelogo">

Продолжение листинга 3.18

<h1>Производство мягкой мебели</h1></div>

</div></div>

<div class="menu" id="gmenu">

<div class="centerall">

<ul><li><a href="/page/main.aspx">Главная</a></li>

<li><a href="/page/products.aspx">Продукция</a></li>

<li><a href="/page/dostavka.aspx">Доставка</a></li>

<li><a href="/page/contact.aspx">Контакты</a></li>

<li><a href="/page/calculator.aspx">Калькулятор</a></li>

<ul class="reg"><li>

<a href="/page/shopcart.aspx">Корзина</a>

</li></ul>

</ul></div></div>

</header>

Листинг 3.18

При переходе на другие страницы, верхняя часть будет оставаться неизменной за счет использования мастер страницы.

Мастер страница имеет возможность размещать в себе теги *asp:ContentPlaceHolder* с уникальными *ID*, за счет чего, страницы, наследующие мастер страницу, смогут размещать в них свой контент по тегу *asp:Content.*

<asp:ContentPlaceHolder ID="title" runat="server">

</asp:ContentPlaceHolder>

<asp:ContentPlaceHolder ID="head" runat="server">

</asp:ContentPlaceHolder>

<asp:ContentPlaceHolder ID="contentDiv" runat="server">

</asp:ContentPlaceHolder>

<asp:Content ID="Content2" ContentPlaceHolderID="head" runat="server">

<link rel="stylesheet" href="../css/main.css" />

</asp:Content>

Листинг 3.19 – теги *asp:ContentPlaceHolder и asp:Content*

Стоит отметить, что в мастер странице подключены библиотеки, представленные в листинге ниже.

<link rel="stylesheet" href="../css/bootstrap.css" />

<link rel="stylesheet" href="../css/jquery-ui.min.css" />

<script src="../scripts/jquery-1.7.2.min.js"></script>

<script src="../scripts/jquery-ui.min.js"></script>

<script src="../scripts/bootstrap.min.js"></script>

<link href="../css/style.css" rel="stylesheet" type="text/css"/>

Листинг 3.20

С помощью тега *link* подключаются таблицы каскадных стилей css, а с помощью тега *script* – *javascript* коды.

Выделение элементов меню, представленное на рисунке 3.2, происходит с помощью кода на *javascript* и *jquery,* указанного в листинге 3.21

$(document).ready(function(){

$('#gmenu a').each(function () {

if($(this).attr('href') == location.pathname) $(this).addClass('Active');

});

});

Листинг 3.21

В этом коде выделенному элементу меню присваивается класс *Active*. Скрипт срабатывает при каждой смене страницы.

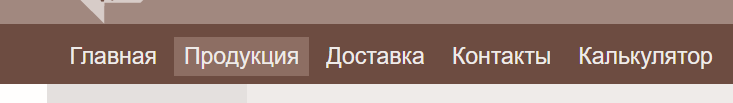


Рисунок 3.2 – Выделение меню

Страница с продукцией представлена на рисунке 3.3.

Страница разделена на две части: поиск и найденные товары.

Код *Html* для реализации поиска представлен в листинге 3.22

<div class="leftsearch">

<p class="searchp">Категория мебели</p>

<select runat="server" class="Category form-control" id="Select" name="Category">

<option value="0" selected="selected">Диваны</option>

<option value="1">Тахты</option>

<option value="2">Кресла</option>

<option value="3">Стулья</option>

<option value="4">Пуфы</option>

<option value="5">Банкетки</option>

<option value="6">Все</option>

</select>

<p class="searchp">Подгатегория</p>

<input id="ExtensionBox" type="text" runat="server" class="leftsearchcontrol form-control"/>

<p class="searchp">Название</p>

Продолжение листинга 3.22

<input id="NameBox" type="text" runat="server" class="leftsearchcontrol form-control"/>

<p class="searchpce">Цена</p>

<p><input type="text" id="FromCost" name="FromCost"/>

<input type="text" id="ToCost" name="ToCost"/>

</p>

<div id="slider-range">

</div>

Листинг 3.22

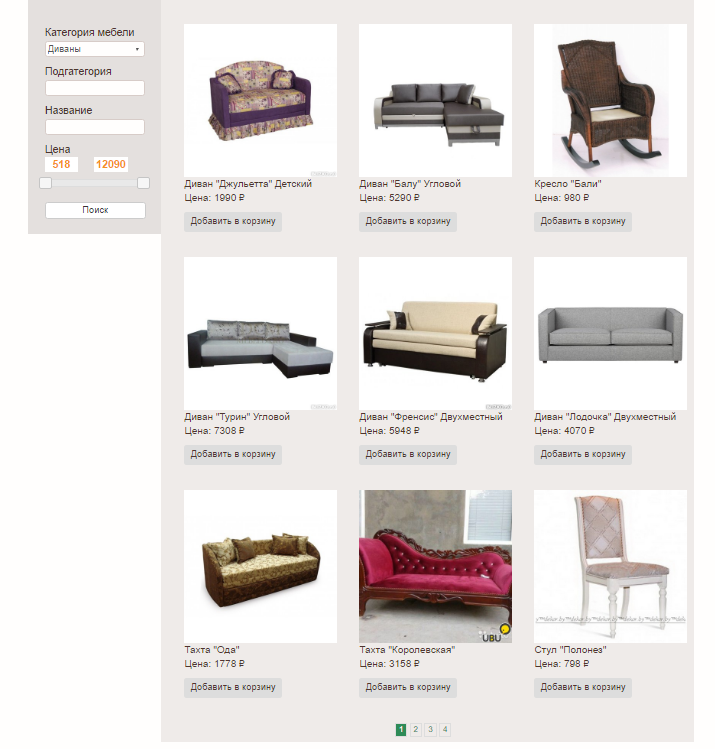


Рисунок 3.3 – Страница продукции

Для реализации ползунков используется элемент *slider* библиотеки *jquery-ui*. Код реализации представлен в листингах 3.23, 3.24.

function costSelect() {

$.ajax({

type: "POST",

url : this.location.href + "/costSelect",

data: '{}',

contentType: "application/json; charset=utf-8",

dataType: "json",

success: cost,

failure: function (response)

{

alert(response.responseText);

},

error: function (response)

{

alert(response.responseText);

}

});}

Листинг 3.23

В этом коде с помощью *jquery* *ajax* загружаются с сервера минимальная и максимальная цена всей продукции в виде файла *XML*. При удачной загрузке этих значений выполняется функция *cost*.

function cost(response) {

var xmlDoc = $.parseXML(response.d);

var xml = $(xmlDoc);

min = parseInt(xml.find("MINCOST").text());

max = parseInt(xml.find("MAXCOST").text());

$("#slider-range").slider({

range: true,

min: min,

max: max,

values: [min, max],

slide: function (event, ui) {

$("#FromCost").val(ui.values[0]);

$("#ToCost").val(ui.values[1]);

}

});

$("#FromCost").val($("#slider-range").slider("values", 0));

$("#ToCost").val($("#slider-range").slider("values", 1));}

Листинг 3.24

Данная функция ищет в *XML* файле минимальное и максимальное значение и записывает их в переменные *min* и *max*. Далее происходит инициализация элемента *slider*. В фигурных скобках указываются пары ключ – значение. Указывается, что *slider* будет иметь два ползунка, минимальным и максимальным значениями будут значения, найденные нами выше, а также указывается с какими элементами будет связан *slider*.

Для показа продукции используется элемент *ListView*. Шаблон для каждого элемента представлен в листинге 3.25.

<ItemTemplate>

<td runat="server" class="width">

<a href='<%#"detailinfo.aspx?Id=" + Eval("Id") %>'>

<asp:Image ID="Image" runat="server"

Width="230px" Height="230px" ImageUrl='<%#Eval("Image") %>'/>

</a>

<br />

<p class="template">

<%#Eval("Category") + " \"" +

Eval("Name") +"\" "+Eval("Extension") +

"<br>" +"Цена: "+ Eval("Cost")+" ₽" %>

</p>

<button name='add' class="add btn .btn-default"

type='submit' value='<%#Eval("Id") %>' >

Добавить в корзину </button>

</td>

</ItemTemplate>

Листинг 3.25

В данном элементе содержится фотография товара и его описания. При нажатии на картинку организуется *GET*-запрос, организующий переход на страницу с подробным описанием товара. Кнопка с именем *add* осуществляет добавление товара в корзину. Ключевое слово *Eval* используется для привязки данных к элементу шаблона *ListView* по ключу.

Перейдя на страницу корзины без добавленных товаров, пользователю будет выведена информация, представленная на рисунке 3.4.

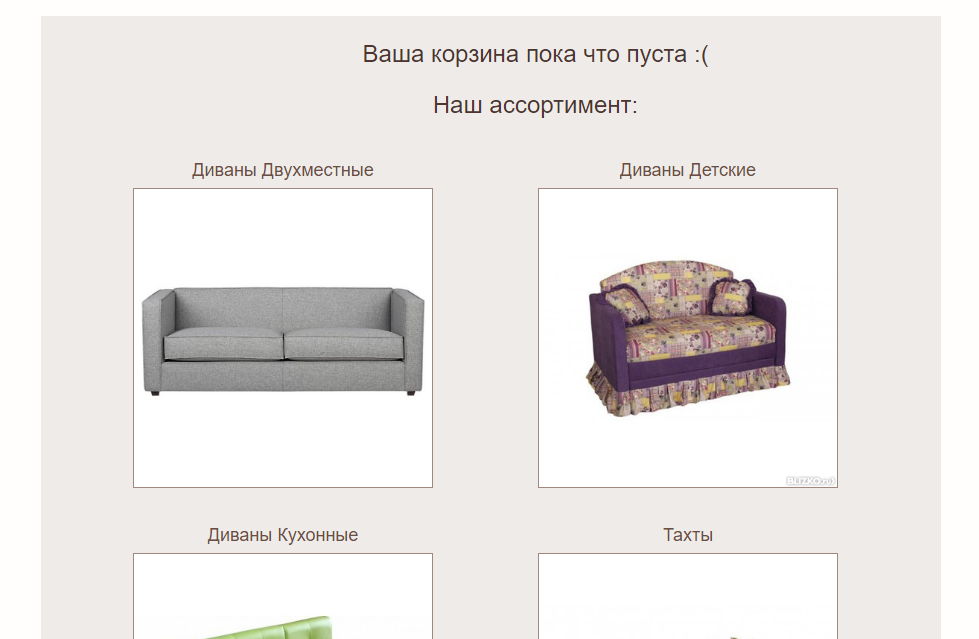


Рисунок 3.4 – Страница с пустой корзиной

На данной странице используется шаблон для пустого источника данных элемента *GridView*. Код *Html* представлен в листинге 3.26.

<EmptyDataTemplate>

<div class="empty">

<h3 class="h31">Ваша корзина пока что пуста :(</h3>

<h3 class="h32">Наш ассортимент:</h3>

<div class="assort">

<ul class="main">

…

</ul>

</div>

</div>

</EmptyDataTemplate>

Листинг 3.26

Если в корзину добавлены товары, то пользователю предоставляется страница с содержимым, представленным на рисунке 3.5

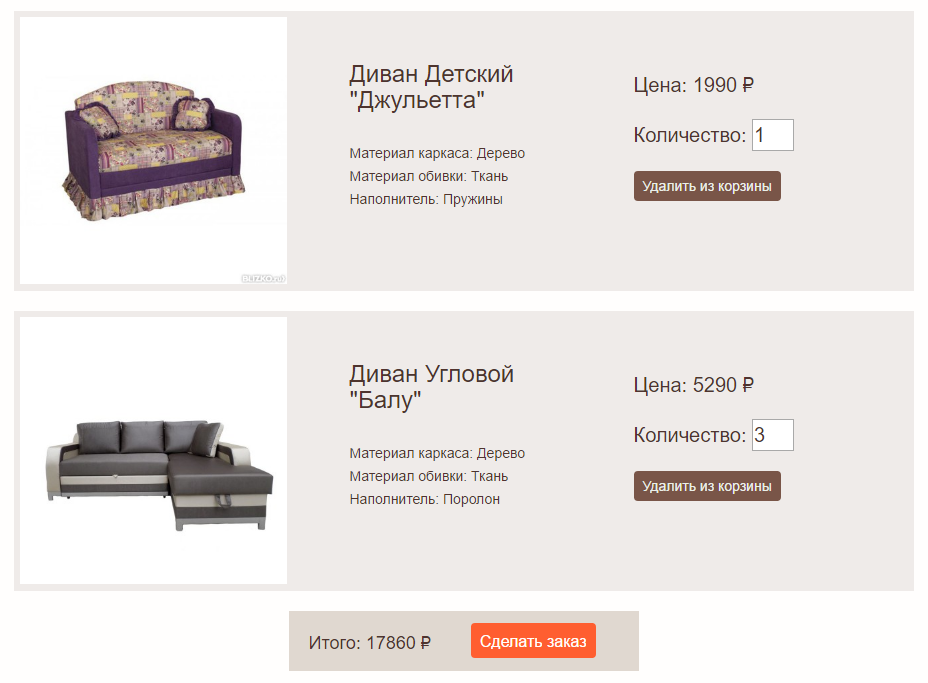


Рисунок 3.5 – Страница наполненной корзины

Для показа каждого элемента используется шаблон, представленный в листинге 3.27.

<ItemTemplate>

<div class="imgitem"><asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl='<%#"detailinfo.aspx?Id=" + Eval("Id") %>' Text="" CssClass="imgitem">

<asp:Image ID="Image1" runat="server"

ImageUrl='<%#Eval("Image") %>' Width="267" />

</asp:HyperLink></div>

<div class="itemshop"><div class="itemshopleft">

<h3 class="h3item"> <%# Eval("Category")+" "+Eval("Extension")+"<br>"+"\""+Eval("Name")+"\"" %> </h3>

<ul><li>Материал каркаса: <%#Eval("Material") %></li>

<li>Материал обивки: <%#Eval("Obivka") %></li>

<li>Наполнитель: <%#Eval("Napolnitel") %></li>

</ul></div>

<div class="itemshopright">

<asp:Label ID="cost" runat="server" CssClass="label1 costText" Text='<%#"Цена: "+Eval("Cost") +" ₽"%>'></asp:Label><br />

<span class="label1 label2">Количество: </span>

<input id="number" class="numberText label1 label2" type="text" runat="server" value='<%#Eval("Number") %>'

Продолжение листинга 3.27

onchange="itogo()"/><br />

<button name='removefcart' class='btn .btn-default' type='submit' value='<%#Eval("Id")%>' formnovalidate="formnovalidate"> Удалить из корзины </button></div></div></ItemTemplate>

<FooterTemplate >

<div class="footer"><asp:Label ID="itogo" runat="server" CssClass="label1 foot1" Text=""></asp:Label>

<input id="Order" type="button" class="order btn foot1" value="Сделать заказ" /> </div></FooterTemplate>

Листинг 3.27

При нажатии на кнопку сделать заказ появляется всплывающее окно, представленное на рисунке 3.6. Окно содержит данные о заказчике, которые будут добавлены в базу данных.

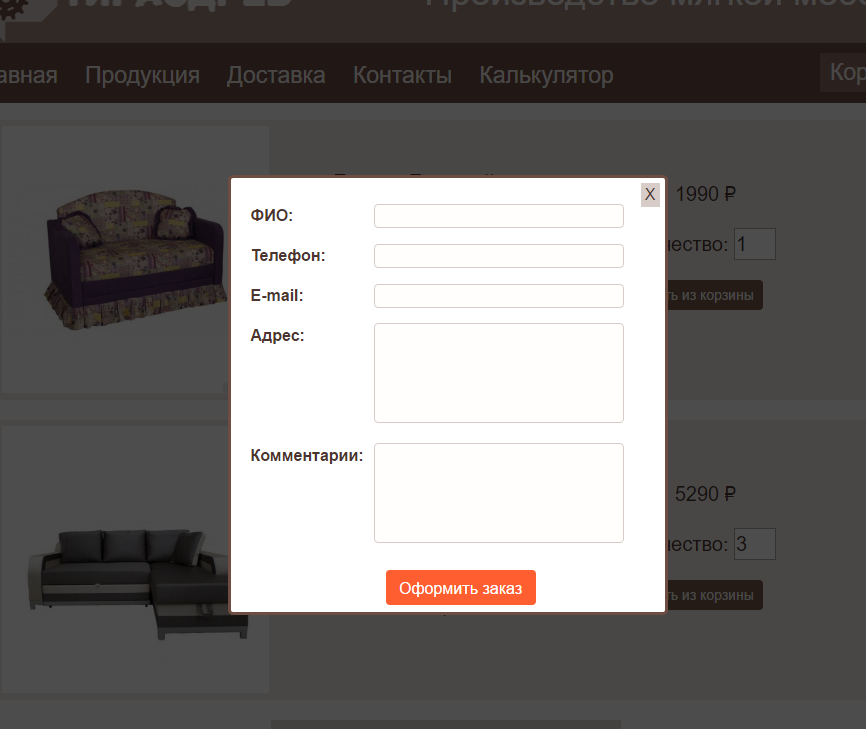


Рисунок 3.5 – Оформление заказа

Данное окно появляется с помощью *javascript* кода, представленного в листинге 3.28.

$(document).ready(function () {

$('.order').click(function () {

$('.overlay').fadeIn(100,

function () {

$('.modal\_form')

.css('display', 'block')

.animate({ opacity: 1, top: '50%' }, 50);

}); });

Листинг 3.28

В данном коде окну присваивается *css* правило, которое делает его видимым для пользователя, а также правило, затемняющее внешнюю область.

На рисунке 3.6 представлена главная форма *Windows*-приложения.

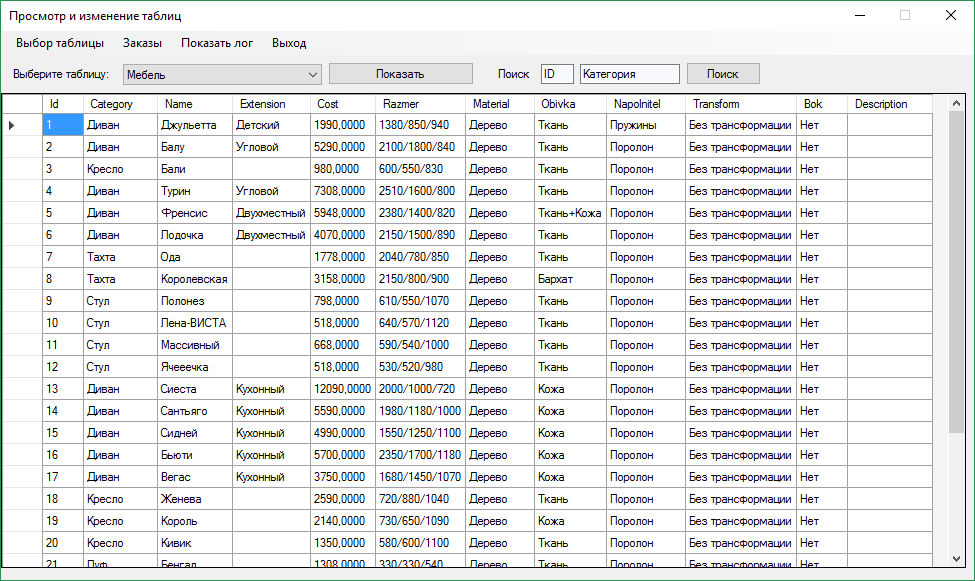


Рисунок 3.6 – Главная страница *Windows*-приложения

В верхней части формы расположено меню, дублирующее некоторые функции приложения. Под меню находится строка работы с *DataGridView*, которая находится ниже. Данная строка включает в себя выбор таблицы, а также поиск по ней.

При нажатии на какую-либо ячейку в *DataGridView*, выпадает контекстное меню, как показано на рисунке 3.7.

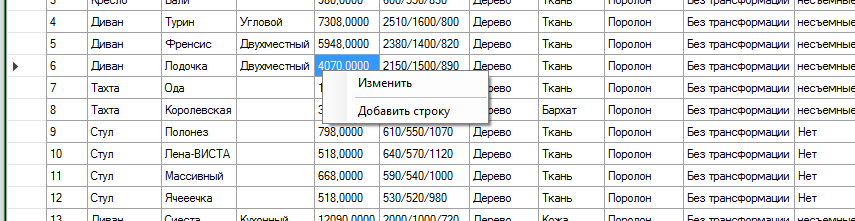


Рисунок 3.6 – Выпадающее меню на главной форме

Контекстное меню содержит два пункта: «Изменить» и «Добавить строку». По нажатию на первый пункт появится окно изменения с данными этой строки, как представлено на рисунке 3.7.

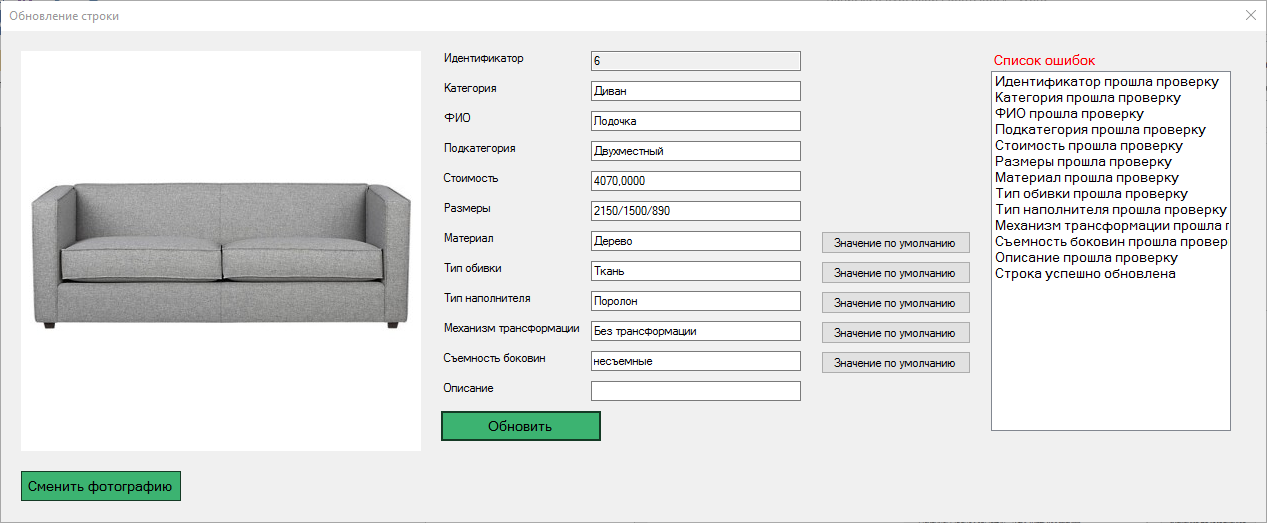


Рисунок 3.7 – Форма для обновления данных

На данной форме в левой части содержится фотография изделия, которую можно изменить, нажав на кнопку «Сменить фотографию» правее от фото находится описание, и справа поле для «Списка ошибок», которое заполняется в зависимости от данных, введенных в поля для заполнения.

Данная форма предназначена для изменения базы данных. Строка таблицы будет обновлена только, если все поля будут корректны.

При нажатии на кнопку добавить строку будет показана аналогична форма, но с незаполненными полями.

Из главной формы доступен переход на форму работы с заказами. Скриншот данной формы представлен на рисунке 3.8.

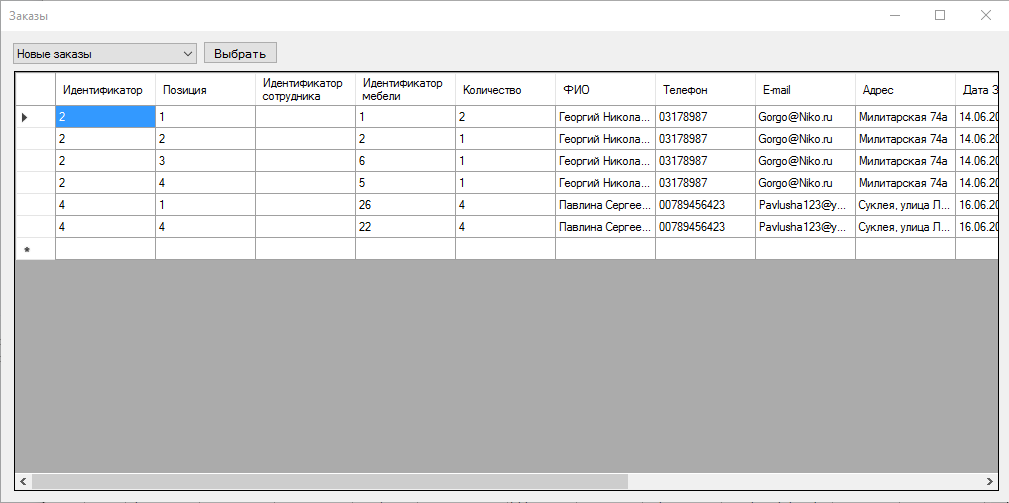


Рисунок 3.8 – Форма для работы с заказами

Работа с заказами включает в себя четыре стадии: новые заказы, на исполнении, ожидающие доставки и доставленные заказы. При нажатии на любую ячейку таблицы будет показано всплывающее меню, представленное на рисунке 3.9.

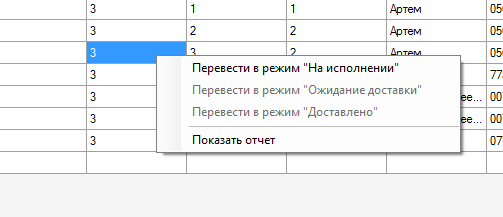


Рисунок 3.9 – Контекстное меню формы «Заказы»

На каждой стадии заказ можно перевести лишь на одну стадию вперед.

3.3 Полученные результаты и их анализы

Результатами работы программы являются выполненные заказы и составленные по ним отчеты. Структура отчета представлена на рисунке 3.10.

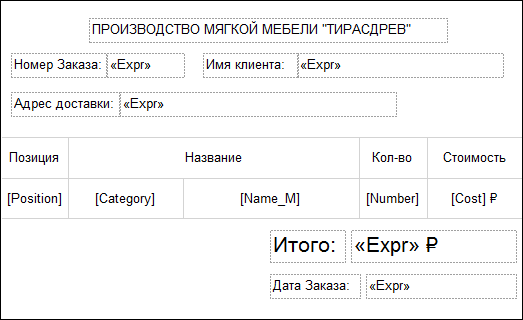


Рисунок 3.10 – Структура отчета

Отчет содержит данные о клиенте предприятия, а также информацию по приобретенным им товарам. Пример отчета представлен на рисунке 3.11.

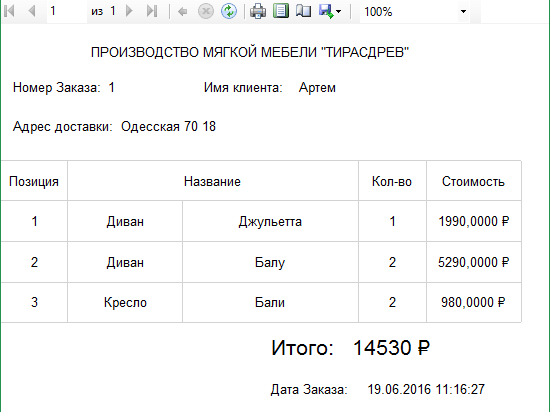


Рисунок 3.11 – Пример отчета

Также результатом программы является логирование действий сотрудников и клиентов. Все операции, касающиеся изменения и добавления данных в базу данных, добавляются в отдельную таблицу. Посмотреть лог действий можно, нажав на пункт меню «Показать лог». Форма, открывающаяся после нажатия данной кнопки представлена на рисунке 3.12.

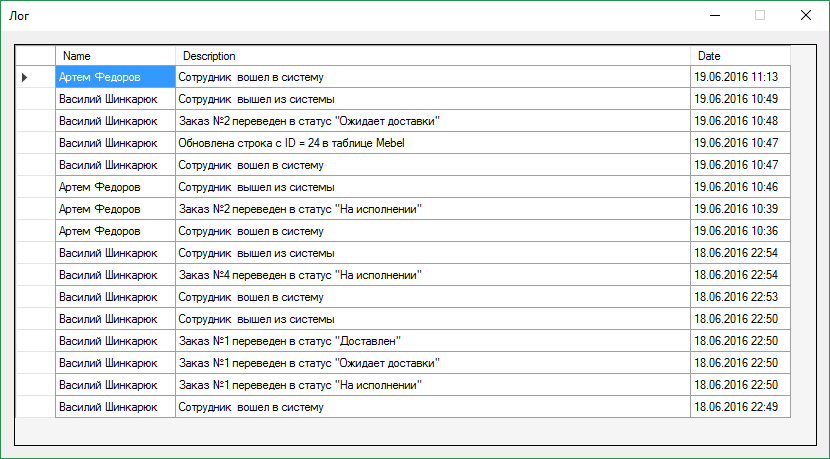


Рисунок 3.12 – Форма логирования

В форме содержатся такие данные как имя сотрудника, описание действия и дата события. Данные выводятся отсортированными по дате в обратном порядке.

**ЗАКЛЮЧЕНИЕ**

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Wikipedia
2. ProfessorWeb
3. ITVDN
4. Stackoverflow