**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
| 1. | Разработка программного комплекса……………………………… | ...3 |
| 1.1. | Разработка уровня доступа к данным……………………………... | ...3 |
| 1.1.1. | Разработка таблиц базы данных…………………………………… | ...3 |
| 1.1.2. | Схематическое представление сущности базы данных………….. | .10 |
| 1.2. | Разработка программной части…………………………………….. | .12 |
| 1.2.1. | Реализация переходов между страницами………………………… | .12 |
| 1.2.2. | Классы ViewModel………………………………………………….. | .12 |
| 1.2.2.1. | Работа с табличными данными…………………………………….. | .12 |
| 1.2.2.2. | Работа с данными для добавления/изменения……………………. | .13 |
| 1.2.3. | Классы сервисы……………………………………………………... | .17 |
| 1.2.4. | Модель базы данных………………………………………………... | .19 |
| 1.2.5. | Работа с библиотекой AutoMapper………………………………… | .21 |
| 1.3. | Разработка пользовательского интерфейса……………………….. | .22 |
| 1.3.1. | Главное окно………………………………………………………… | .22 |
| 1.3.2. | Страницы с информацией………………………………………….. | .23 |
| 1.3.3. | Окна добавления/изменения……………………………………….. | .24 |
| 2. | Руководство пользователя………………………………………….. | .28 |
| 2.1. | Главное окно………………………………………………………… | .28 |
| 2.2. | Страница продукции………………………………………………... | .28 |
| 2.2.1. | Таблица продукции…………………………………………………. | .29 |
| 2.2.2. | Таблица изменений, данных о изделиях…………………………... | .29 |
| 2.2.3. | Окно изменения/добавления изделий……………………………... | .30 |
| 2.3. | Страница материалов……………………………………………….. | .30 |
| 2.3.1. | Таблица материалов……………………………………………….... | .31 |
| 2.3.2. | Окно изменения/добавления материала…………………………… | .32 |
| 2.4. | Страница клиентов………………………………………………….. | .33 |
| 2.4.1. | Окно с контактными данными……………………………………... | .33 |
| 2.4.2. | Таблица всех заказов………………………………………………... | .34 |
| 2.4.3. | Таблица изменения/добавления заказа……………………………. | .34 |
| 2.4.4. | Окно изменения/добавления контактной информации…………... | .35 |
| 2.5. | Страница поставщиков……………………………………………... | .35 |
| 2.5.1. | Страница поставок………………………………………………….. | .36 |
| 2.5.2. | Окно изменения/добавления информации о поставке…………… | .36 |
| 2.6. | Страница персонала………………………………………………… | .37 |
| 2.6.1. | Окно с контактными данными сотрудника…………………....….. | .37 |
| 2.6.2. | Окно изменения/добавления данных о сотруднике………………. | .38 |
|  | Литература…………………………………………………………... | .40 |

1. **РАЗРАБОТКА ПРОГРАММНОГО КОМПЛЕКСА**
   1. **РАЗРАБОТКА УРОВНЯ ДОСТУПА К ДАННЫМ**
      1. **Разработка уровня доступа к данным**

База данных называется: «DaVaNiDataBase».

Название таблицы: **Customers**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | Id | Int |
|  | Name | Nvarchar(100) |
|  | Country | Nvarchar(100) |
|  | City | Nvarchar(100) |
|  | Email | Nvarchar(100) |
|  | Skype | Nvarchar(100) |
|  | Deleted | Bit |

*Id* – идентификатор клиента, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*Name* – имя клиента, не разрешено значение null.

*Country* – страна клиента, не разрешено значение null.

*City* – город клиента, не разрешено значение null.

*Email* – электронная почта клиента, не разрешено значение null.

*Skype* –skype клиента, не разрешено значение null.

*Deleted* – пометка поля которая означает что поле удалено, значение по умолчанию 0 (0 – поле не удалено, 1 – поле удалено).

Таблица **Customer** должна содержать информацию о клиента.

Название таблицы: **Orders**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | Id | Int |
| FK | CustomerId | Int |
|  | OrderNumber | Nvarchar(100) |
|  | Model | Nvarchar(100) |
|  | Quantity | Int |
|  | PriceForOne | Real |
|  | PriceForAll | Real |
|  | OrderDate | DateTime2 |
|  | Deleted | Bit |

*Id* – идентификатор заказа, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*CustomerId* – идентификатор клиента, внешний ключ к таблице [Customers] к столбцу [Id].

*OrderNumber* – номер текущего заказа, не разрешено значение null.

*Model* – модель изделия, не разрешено значение null.

*Quantity* – количество изделий данной модели в текущем заказе, не разрешено значение null.

*PriceForOne* – цена за единицу изделия данной модели, не разрешено значение null.

*PriceForAll* – цена за все изделия данной модели, не разрешено значение null.

*OrderDate* – дата создания заказа, не разрешено значение null.

*Deleted* – пометка поля которая означает что поле удалено, значение по умолчанию 0 (0 – поле не удалено, 1 – поле удалено).

Таблица **Orders** дожна содержать информацию о заказах, и имеет связь один к многим с таблицей **Customers**, которая означает что один клиент может иметь много заказов.

Название таблицы: **Suppliers**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | Id | Int |
|  | Name | Nvarchar(100) |
|  | Country | Nvarchar(100) |
|  | City | Nvarchar(100) |
|  | Email | Nvarchar(100) |
|  | Skype | Nvarchar(100) |
|  | Deleted | Bit |

*Id* – идентификатор поставщика, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*Name* – имя поставщика, не разрешено значение null.

*Country* – страна поставщика, не разрешено значение null.

*City* – город поставщика, не разрешено значение null.

*Email* – электронная почта поставщика, не разрешено значение null.

*Skype* –skype поставщика, не разрешено значение null.

*Deleted* – пометка поля которая означает что поле удалено, значение по умолчанию 0 (0 – поле не удалено, 1 – поле удалено).

Таблица **Suppliers** должна содержать информацию о поставщиках.

Название таблицы: **Supplies**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | Id | Int |
| PK, FK | SupplierId | Int |
|  | MaterialName | Nvarchar(100) |
|  | Description | Nvarchar(500) |
|  | Price | Real |
|  | Deleted | Bit |

*(Id, SupplierId) –* составной ключ.

*Id* – идентификатор поставки, авто инкремент (начало = 1, шаг = 1.

*SupplierId* – идентификатор поставщика, внешний ключ к таблице [Suppliers] к столбцу [Id].

*MaterialName* – название материала поставки, не разрешено значение null.

*Description* – дополнительная информация о заказе, не разрешено значение null.

*Price* – цена материала данной поставки, не разрешено значение null.

*Deleted* – пометка поля которая означает что поле удалено, значение по умолчанию 0 (0 – поле не удалено, 1 – поле удалено).

Таблица **Supplies** должна содержать информацию о поставках, и имеет связь один к многим с таблицей **Suppliers,** которая означает что один поставщик может иметь много поставок.

Название таблицы: **Employees**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | Id | Int |
|  | Name | Nvarchar(100) |
|  | EmploymentDate | DateTime2 |
|  | Post | Nvarchar(100) |
|  | Salary | Int |
|  | Deleted | Bit |

*Id* – идентификатор сотрудника, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*Name* – имя сотрудника, не разрешено значение null.

*EmploymentDate* – дата приема на работу, не разрешено значение null.

*Post* – должность сотрудника, не разрешено значение null.

*Salary* – заработная плата сотрудника, не разрешено значение null.

*Deleted* – пометка поля которая означает что поле удалено, значение по умолчанию 0 (0 – поле не удалено, 1 – поле удалено).

Таблица **Employees** дожна содержать информацию о сотрудниках.

Название таблицы: **Photos**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | Id | Int |
| FK | EmployeeId | Int |
|  | Name | Nvarchar(100) |
|  | DataBytes | Varbinary(max) |
|  | Deleted | Bit |

*Id* – идентификатор документа, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*EmployeeId* – идентификатор сотрудника, внешний ключ к таблице [Employees] к столбцу [Id].

*Name* – название документа, не разрешено значение null.

*DataBytes* – бинарный код данного изображения, не разрешено значение null.

*Deleted* – пометка поля которая означает что поле удалено, значение по умолчанию 0 (0 – поле не удалено, 1 – поле удалено).

Таблица **Photos** должна содержать информацию о документах сотрудника, и имеет связь один к многим с таблицей **Employees**, означает что один сотрудник может иметь много электронных копий документов.

Название таблицы: **TelephoneNumbers**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | Id | Int |
| FK | CustomerId | Int |
| FK | EmployeeId | Int |
| FK | SupplierId | Int |
|  | TelNumber | Nvarchar(20) |

*Id* – идентификатор телефонного номера, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*CustomerId* – идентификатор клиента, внешний ключ к таблице [Customers] к столбцу [Id], разрешено значение null.

*EmployeeId* – идентификатор сотрудника, внешний ключ к таблице [Employees] к столбцу [Id], разрешено значение null.

*SupplierId* – идентификатор поставщика, внешний ключ к таблице [Suppliers] к столбцу [Id], разрешено значение null.

*TelNumber* – номер телефона, не разрешено значение null.

Таблица **TelephoneNumbers** должна содержать телефонные номера, и имеет три связи один к многим с таблицами **Employees, Customers** and **Suppliers** tables.

* Первая связь означает, что один сотрудник может иметь много номеров телефона.
* Вторая связь означает, что один клиент может иметь много номеров телефона.
* Третья связь означает, что один поставщик может иметь много номеров телефона.

Все эти связи являются не обязательными, так как один номер телефона не может принадлежать нескольким людям одновременно.

Название таблицы: **Products**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | Id | Int |
|  | Type | Nvarchar(100) |
|  | ModelName | Nvarchar(100) |
|  | M | Int |
|  | L | Int |
|  | XL | Int |
|  | XXL | Int |
|  | Material1 | Nvarchar(100) |
|  | Material2 | Nvarchar(100) |
|  | Price | Real |
|  | Deleted | Bit |

*Id* – идентификатор продукта, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*Type* – тип материала, не разрешено значение null.

*ModelName* – название изделия, не разрешено значение null.

*M* – количество изделий с размером M, не разрешено значение null.

L – количество изделий с размером L, не разрешено значение null.

*XL* – количество изделий с размером XL, не разрешено значение null.

*XXL* – количество изделий с размером XXL, не разрешено значение null.

*Material1* – первый внешний материал изделия, не разрешено значение null.

*Material2* – второй внешний материал изделия, не разрешено значение null.

*Price* – цена данного изделия, не разрешено значение null.

*Deleted* – пометка поля которая означает что поле удалено, значение по умолчанию 0 (0 – поле не удалено, 1 – поле удалено).

Таблица **Products** должна содержать информацию об всех изделиях.

Название таблицы: **InformationChanges**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | Id | Int |
| PK, FK | ProductId | Int |
|  | DateOfChanges | DateTime2 |
|  | DescriptionBefore | Nvarchar(1000) |
|  | DescriptionAfter | Nvarchar(1000) |

*(Id, ProductId) –* составной ключ.

*Id* – идентификатор информационного поля, авто инкремент (начало = 1, шаг = 1.

*ProductId* – идентификатор изделия, внешний ключ к таблице [Products] к столбцу [Id].

*DateOfChanges* – дата изменения, не разрешено значение null.

*DescriptionBefore* – информация об изделии до изменения, разрешено значение null.

*DescriptionAfter* – информация об изделии после изменения, не разрешено значение null.

Таблица **InformationChanges** должна содержать информацию о изменениях, произошедших с изделием, и имеет связь один к многим с таблицей **Products**, означает что одно изделие может иметь много изменений.

Название таблицы: **MatType**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | MatTypeId | Int |
|  | MatTypeName | Nvarchar(100) |

*MatTypeId* – идентификатор типа материала, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*MatTypeName* – название типа материала, не разрешено значение null.

Таблица **MatType** должна содержать информацию о типах материалов. Данные в таблицу добавляются автоматически при создании базы данных.

Название таблицы: **ParamType**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | ParamTypeId | Int |
|  | ParamTypeProp | Nvarchar(100) |

*ParamTypeId* – идентификатор параметра, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*ParamTypeProp* – название параметра типа материала, не разрешено значение null.

Таблица **ParamType** должна содержать информацию о параметрах типов материалов. Данные в таблицу добавляются автоматически при создании базы данных.

Название таблицы: **MatParamType**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK, FK | MatTypeId | Int |
| PK, FK | ParamTypeId | Int |

*(MatTypeId, ParamTypeId) –* составной ключ.

*MatTypeId* – идентификатор типа материала, внешний ключ к таблице [MatType] к столбцу [MatTypeId].

*ParamTypeId* – идентификатор параметра, внешний ключ к таблице [ParamType] к столбцу [ParamTypeId].

Таблица **MatParamType** должна содержать информацию о том какие параметры отвечают каждому типу материалов. Данные в таблицу добавляются автоматически при создании базы данных.

Название таблицы: **Material**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK | MaterialId | Int |
| FK | MatTypeId | Int |
|  | Deleted | Bit |

*MaterialId* – идентификатор материала, авто инкремент (начало = 1, шаг = 1), первичный ключ.

*MatTypeId* – идентификатор типа материала, внешний ключ к таблице [MatType] к столбцу [MatTypeId].

*Deleted* – пометка поля которая означает что поле удалено, значение по умолчанию 0 (0 – поле не удалено, 1 – поле удалено).

Таблица **Material** нужна для возможности создания разных материалов одного типа, а также имеет связь один к многим с таблицей **MatType**.

Название таблицы: **MaterialParams**

|  |  |  |
| --- | --- | --- |
| ***Тип поля*** | ***Название столбца*** | ***Тип данных*** |
| PK, FK | MaterialId | Int |
| PK, FK | MatTypeId | Int |
| PK, FK | ParamTypeId | Int |
|  | Value | Nvarchar(100) |

*(MaterialId, MatTypeId, ParamTypeId) –* составной ключ.

*MaterialId* – идентификатор материала, внешний ключ к таблице [Material] к столбцу [MaterialId].

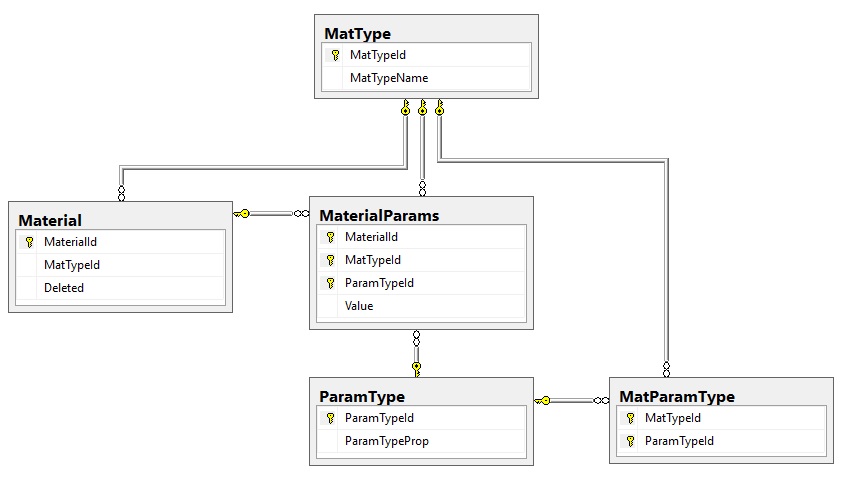
*MatTypeId* – идентификатор типа материала, внешний ключ к таблице [MatParamType] к столбцу [MatTypeId].

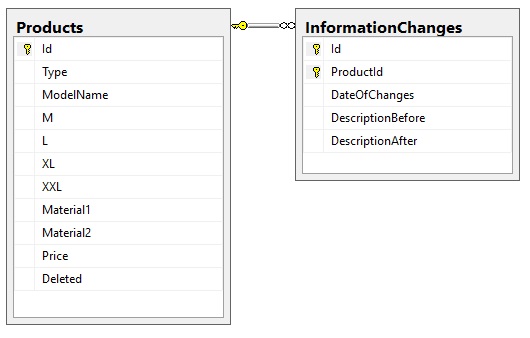
*ParamTypeId* – идентификатор параметра, внешний ключ к таблице [MatParamType] к столбцу [ParamTypeId].

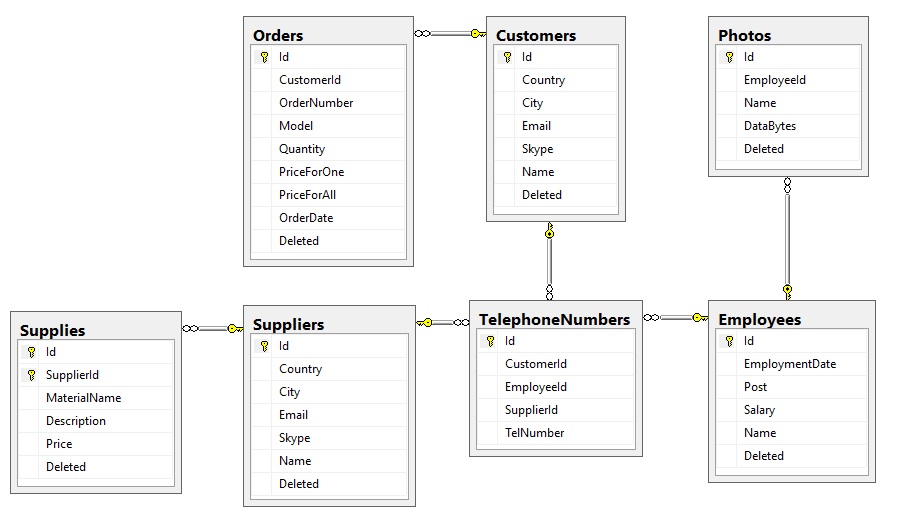
*Value* – значение параметра, не разрешено значение null.

Таблица **MaterialParams** должна содержать значение пареметров патериала с идентификатором MaterialId, и имеет три связи один к многим с таблицами **MatParamType** и **Material**. Это дает возможность создать материал со всеми параметрами для определенного типа.

* + 1. **Схематическое представление сущности базы данных**

****



****

* 1. **РАЗРАБОТКА ПРОГРАММНОЙ ЧАСТИ**
     1. **Реализация переходов между страницами**

Первое что мы встречает в программе это переходи со страницы на страницу при помощи элемента <Frame/> который имеет название «MainAreaFrame», название этому элементу присваивается для возможности обратится к нему из кода, что мы и делаем. Для переходов используется класс «NavigationService» из пространства имен «System.Windows.Navigation», а именно его метод «Navigate» который позволяет совершать переходы по страницам. В программе используется два типа переходов: первый, путем сообщения методу «Navigate» относительного пути к нужной странице, а второй путем создания экземпляра страницы.

MainAreaFrame.NavigationService.Navigate

(new Uri("Pages/Products/ProductPage.xaml", UriKind.Relative));

NavigationService.Navigate(new SupplyInformationPage());

* + 1. **Классы ViewModel**
       1. **Работа с табличными данными**

Речь пойдет о специальных классах, которые поверхностно уже затрагивали при разборе создания пользовательского интерфейса. Это классы, к которым идет привязка информационных полей из интерфейса, они содержат типизированный массив ObservableCollection<Т>, экземпляр класса, который хранит в себе выделенный объект из таблицы, а также методы с помощью которых идет управление данными.

public ObservableCollection<SheepProductTable> Items { get; set; }

public SheepProductTable SelectedItem { get; set; }

public void Delete(){...}

Класс «SheepProductTable» это класс свойства, которого используются для непосредственной привязки к информационным полям интерфейса, частичная реализация данного класса представлена ниже:

public class SheepProductTable : INotifyPropertyChanged

{

private int \_id;

. . . . .

private float \_price;

public int Id

{

get { return \_id; }

set

{

if (value == \_id) return;

\_id = value;

OnPropertyChanged();

}

}

. . . . . .

public event PropertyChangedEventHandler PropertyChanged;

[NotifyPropertyChangedInvocator]

protected virtual void OnPropertyChanged([CallerMemberName]

string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

Интерфейс «INotifyPropertyChanged» используется для того чтобы в случае программного изменения любого свойства эти изменения переносились на соответствующее поле пользовательского интерфейса.

Реализация заполнения массива «Items» данными выполняется получением данных из базы данных «DaVaNiDataBase» которая создаться при помощи технологии «EntityFramework» (будет рассмотрена дальше в этой главе, пункт 4), а связь между классами «EntityFramework» и «ViewModel» создаться при помощи «AutoMapper», который дает возможность создать связь между полями класса для автоматического копирования данных из одного класса в другой, и в случае надобности есть возможность создания обратной связи (примеры мапингов будут продемонстрированы дальше в этой главе, пункт 5). Пример заполнения приведен ниже:

public SheepProductTableVM()

{

Items = new ObservableCollection<SheepProductTable>(

context.Products.Where(e => !e.Deleted && e.Type == "Sheep")

.ProjectTo<SheepProductTable>(App.Mapper.ConfigurationProvider));

}

Также есть еще один вариант инициализации массива, при помощи использования созданных классов «…Service», в которых происходит манипуляция с данными полученных, как из базы данных, так и из пользовательского интерфейса (сервисы будут рассмотрены далее в этой главе, пункт 3)

Items = new ObservableCollection<LeatherTable>(\_service.GetLeatherMaterials());

* + - 1. **Работа с данными для добавления/изменения**

В окнах для добавления/изменения не нужно загружать всю таблицу данных, достаточно все лишь передать в него идентификатор нужного поля и вытянуть его из базы данных отдельно:

public AddEditEmployeeVM(int id)

{

\_service = new AddEditEmployeeService(App.Mapper.ConfigurationProvider);

if (id == 0)

{

TelephoneNumbers = new ObservableCollection<Telephone>(new[]

{ new Telephone { TelNumber = "" } });

EmploymentDate = DateTime.Now;

Documents = new ObservableCollection<Document>();

}

else

GetEmployeeById(id);

}

На примере видно, что есть два варианта создания окна: первый это добавление, в котором мы инициализируем коллекции, а также те поля стандартные значения которых нас не устраивают. Второй вариант – это изменение где мы вызываем метод, который загружает нужный нам объект.

Коллекции «TelephoneNumbers» и «Documents» это типизированные коллекции, экземпляры классов которых приведены ниже:

public class Telephone

{

public int Id { get; set; }

public int? EmployeeId { get; set; }

public string TelNumber { get; set; }

}

public class Document

{

public int Id { get; set; }

public int EmployeeId { get; set; }

public byte[] Data { get; set; }

public string Name { get; set; }

}

Метод «GetEmployeeById» это метод который получает нужный нам объект из базы данных, используя класс «…Service»:

public void GetEmployeeById(int id)

{

var employee = \_service.GetById(id);

Id = employee.Id;

Name = employee.Name;

TelephoneNumbers =

new ObservableCollection<Telephone>(employee.TelephoneNumbers);

EmploymentDate = employee.EmploymentDate;

Post = employee.Post;

Salary = employee.Salary;

Documents = new ObservableCollection<Document>(employee.Documents);

}

Для того что бы загруженное изображение документа, можно было хранить в базе данных его нужно преобразовать в массив бит «byte[]», для этого используется специальный класс «BitmapToByteConverter» реализация которого представлена ниже:

class BitmapToByteConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter,

CultureInfo culture)

{

MemoryStream stream = new MemoryStream((byte[])value);

BitmapImage image = new BitmapImage();

image.BeginInit();

image.StreamSource = stream;

image.EndInit();

return image;

}

public object ConvertBack(object value, Type targetType, object parameter,

CultureInfo culture)

{

Stream stream = ((BitmapImage)value).StreamSource;

byte[] buffer = null;

if (stream != null && stream.Length > 0)

{

using (BinaryReader br = new BinaryReader(stream))

{

buffer = br.ReadBytes((int)stream.Length);

}

}

return buffer;

}

}

При рассмотрении пользовательского интерфейса была сказано, что поля в окно добавления/изменения создаются автоматически. Существуют классы, унаследованные от интерфейса «IParamValue»:

public interface IParamValue

{

ParamTypeId ParametrId { get; set; }

string ParametrName { get; set; }

string ParametrValue { get; set; }

}

Существует три класса унаследованных от данного интерфейса: «TextParamValue» для текстовых полей, «DropdownParamValue» для выпадающего списка и «RadiobuttonParamValue» для радиокнопок. У последних двух имеются коллекции строковых значений, которые должны в них быть.

Класс, который отвечает за от какой шаблон элемента будет создан в пользовательском интерфейсе называется «TemplateSelector», в котором реализован всего один метод «SelectTemplate», реализация которого представлена ниже:

public override DataTemplate SelectTemplate(object item, DependencyObject container)

{

FrameworkElement element = container as FrameworkElement;

if (element != null && item != null && item is IParamValue)

{

if (item is TextParamValue)

return element.FindResource("TextTemplate") as DataTemplate;

if (item is DropdownParamValue)

return element.FindResource("DropdownTemplate") as DataTemplate;

if (item is RadiobuttonParamValue)

return element.FindResource("RadiobuttonTemplate") as DataTemplate;

}

return null;

}

Последним в этапе динамического создания шаблонов полей нам нужно знать от чего зависит то какое поле будет создано. Поля создаются в зависимости от того какой тип материала выбран, например, для поля с сырьем и поставщиками – это выпадающий список, для типа материала это радиокнопка, а для всех остальных это обычное текстовое поле. Код реализации представлен ниже:

private IParamValue GetParamValue(ParamTypeId paramTypeId, MatTypeId materialTypeId)

{

IParamValue value = null;

switch (paramTypeId)

{

case ParamTypeId.Raw:

{

if ((materialTypeId == MatTypeId.Leather) ||

(materialTypeId == MatTypeId.Fur))

{

value = new DropdownParamValue(new[]

{ "Овчина", "Телятина", "Дестрой" });

}

if (materialTypeId == MatTypeId.Textile)

{

value = new DropdownParamValue(new[] { "Плащевка", "Шерсть" });

}

break;

}

case ParamTypeId.Type:

value = new RadiobuttonParamValue

(new[] {"Искуственный", "Натуральный"});

break;

case ParamTypeId.Supplier:

value = new DropdownParamValue

(context.Suppliers.Select(e => e.Name).ToList());

break;

default:

value = new TextParamValue();

break;

}

return value;

}

Теперь зная, как создаются шаблоны полей вернемся к нашему «ViewModel» для того что бы узнать, как же идет загрузка информации в окно добавления/изменения материалов.

При создании объекта класса «AddEditMaterialVM\_new», для добавления не нужно ничего передавать, а для редактирования определенного материала нудно передать в конструктор его идентификатор. В конструкторе данного класса мы загружаем названия типов материалов для нашего выпадающего списка, в зависимости от выбора в котором будет создаваться поля, а также идет проверка, был ли передан идентификатор и если он был передан, то мы получаем необходимый нам материал из базы данных, в выпадающем списке мы выбираем тип данного материала и в конце подгружаем данные во все поля.

public AddEditMaterialVM\_new(int id)

{

\_materialsService = new MaterialsService();

MaterialNames =

new ObservableCollection<MatName>(\_materialsService.GetMaterialNames());

if (id != 0)

{

var item = context.Material.Single(e => e.MaterialId == id);

SelectedMaterial = MaterialNames.SingleOrDefault

(e => e.Id == item.MatTypeId);

ParamValues = new ObservableCollection<IParamValue>

(\_materialsService.GetParamsForMaterial(id));

}

}

Благодаря тому, что данный класс реализует интерфейс «INotifyPropertyChanged», мы можем подписать нашу коллекцию параметров на изменение выбранного типа материала, что дает возможность автоматизировать загрузку соответствующих параметров для выбранного типа материала.

public MatName SelectedMaterial

{

get { return \_selectedMaterial ?? (SelectedMaterial = MaterialNames.First()); }

set

{

if (Equals(value, \_selectedMaterial)) return;

\_selectedMaterial = value;

ReloadParamValues();

OnPropertyChanged();

}

}

private void ReloadParamValues()

{

ParamValues = new ObservableCollection<IParamValue>

(\_materialsService.GetParamsForNewMaterial(SelectedMaterial.Id));

}

Для добавлений и изменения материала приведенный ниже методы:

public void AddMaterial()

{

\_materialsService.AddMaterial(ParamValues, SelectedMaterial.Id);

}

public void EditMaterial(int materialId)

{

foreach (var value in ParamValues)

{

\_materialsService.UpdateParamValue(materialId, value);

}

}

* + 1. **Классы сервисы**

Как вы могли заметить в большинстве приведённых выше методов используются объекты классов «…Service». Они в основном используются для окон добавления и изменения, но также в данной программе они еще используются для таблиц материалов, так как материалы хранятся в нескольких таблицах базы данных (подробнее пункт 4 данной главы). Один из методов получения набора материалов приведен ниже:

public IEnumerable<TextileTable> GetTextileMaterials()

{

var materialList = context.Material.Where

(e => e.MatTypeId == MatTypeId.Textile && !e.Deleted);

foreach (var material in materialList)

{

material.MaterialParams =

new List<MaterialParams>(context.MaterialParams.Where

(e => e.MaterialId == material.MaterialId));

}

return materialList.Select(material => new TextileTable

{

Id = material.MaterialId,

Number = material.MaterialParams.FirstOrDefault

(e => e.ParamTypeId == ParamTypeId.Number).Value,

MaterialName = material.MaterialParams.FirstOrDefault

(e => e.ParamTypeId == ParamTypeId.MaterialName).Value,

Raw = material.MaterialParams.FirstOrDefault

(e => e.ParamTypeId == ParamTypeId.Raw).Value,

Color = material.MaterialParams.FirstOrDefault

(e => e.ParamTypeId == ParamTypeId.Color).Value,

Quality = material.MaterialParams.FirstOrDefault

(e => e.ParamTypeId == ParamTypeId.Quality).Value,

Quantity = material.MaterialParams.FirstOrDefault

(e => e.ParamTypeId == ParamTypeId.Quantity).Value,

Size = material.MaterialParams.FirstOrDefault

(e => e.ParamTypeId == ParamTypeId.Size).Value,

Price = material.MaterialParams.FirstOrDefault

(e => e.ParamTypeId == ParamTypeId.Price).Value,

Supplier = material.MaterialParams.FirstOrDefault

(e => e.ParamTypeId == ParamTypeId.Supplier).Value

});

}

В данном методе мы сначала загружаем коллекцию материалов, которая отвечает нашим требования, дальше для каждого материала загружаем коллекцию параметров, и в конце превращаем таблицу базы данных в необходимый нам класс-представление таблицы материалов.

Для окон добавления и изменения сервисы выполняют основные действия: получение определенного объекта по идентификатору, обновление, добавление и удаление объектов из базы данных.

Метод для получение нужного объекта принимает в качестве параметра идентификатор объекта, после чего при помощи LinQ запросы мы получаем этот объект из БД. После того как у нас есть объект, в данном случае мы получаем данные поставщика, мы загружаем его номера телефонов и добавляем их в данные объект. Код реализующий получение приведен ниже:

public AddEditSupplierContactsVM GetById(int id)

{

AddEditSupplierContactsVM supp = (from suppl in context.Suppliers

where suppl.Id == id && !suppl.Deleted

select new AddEditSupplierContactsVM

{

Id = suppl.Id,

Name = suppl.Name,

Country = suppl.Country,

City = suppl.City,

Email = suppl.Email,

Skype = suppl.Skype

}).First();

var telephons = context.TelephoneNumbers.Where(e => e.SupplierId == id)

.ProjectToList<AddEditSupplierContactsVM.Telephone>(\_provider);

supp.TelephoneNumbers =

new ObservableCollection<AddEditSupplierContactsVM.Telephone>();

foreach (var item in telephons)

{

supp.TelephoneNumbers.Add(item);

}

return supp;

}

Обновление объекта происходит схожим путем. Метод в качестве аргумента принимает объект, который мы хотим обновить, дальше получает устаревший объект из базы данных сравнивает все поля и в случае их отличия перезаписывает значение. Дальше если объект имеет некоторые коллекции объектов идет проверка каждого элемента из коллекции и элементами из БД и при отличии их отличии перезаписывает значение, а в случае его отсутствии – объект добавляется. В конце сохраняются все изменения в БД.

public void Update(AddEditSupplierContactsVM item)

{

var dbItem = context.Suppliers.SingleOrDefault(e => e.Id == item.Id);

var convItem = App.Mapper.Map<Supplier>(item);

convItem.CopyWithChecking(dbItem,

e => e.Name,

e => e.Country,

e => e.City,

e => e.Email,

e => e.Skype);

foreach (var number in dbItem.TelephoneNumbers)

{

if (convItem.TelephoneNumbers.Contains(number))

{

var curNum = convItem.TelephoneNumbers.ToList()

.Find(e => e.Id == number.Id);

number.CopyWithChecking(curNum, e => e.SupplierId, e => e.TelNumber);

}

else

convItem.TelephoneNumbers.Add(number);

}

context.SaveChanges();

}

Метод добавления в качестве аргумента принимает объект, который мы хотим добавить и возвращает тот же объект только уже с идентификатором, полученным из базы данных. Данный метод преобразовывает полученный объект в объект базы данных при помощи автомапинга свойств.

public AddEditSupplierContactsVM Insert(AddEditSupplierContactsVM item)

{

var dbItem = App.Mapper.Map<Supplier>(item);

context.Suppliers.Add(dbItem);

context.SaveChanges();

item.Id = dbItem.Id;

return item;

}

Метод удаления в качестве аргумента принимает идентификатор объекта, а также возвращает удаленный объект. В данном методе мы не удаляем объект из базы данных, а только помечаем его как удаленный, что бы в случае надобности можно было получить удаленные объекты.

public AddEditSupplierContactsVM Delete(int id)

{

var item = GetById(id);

var dbItem = context.Suppliers.Single(e => e.Id == id);

dbItem.Deleted = true;

context.SaveChanges();

return item;

}

* + 1. **Модель базы данных**

Создание базы данных выполнялось при помощи технологии «EntityFramework» с использованием «Code First», который позволять определять модель базы данных при помощи классов C#. Настройка выполнялась при помощи атрибутов для классов и свойств, а также при помощи «Fluent API». Все классы созданные в данной модели практически полностью идентичны классам из «ViewModel» с небольшим отличием в виде свойств типа bool которые предназначены для пометки поля удаленным.

Создание данной модели начинается с создания класса, унаследованного от «DbContext», конструктор которого наследуется от конструктора базового класса и в него передается строковое название строки подключения к базе данных. Так как наша база данных для корректной работы требует изначально заполненных таблиц с типами материалов и параметров, мы используем инициализатор базы данных, который представлен в виде класса у наследованного от «CreateDatabaseIfNotExists <DaVaNiDataBase>» в котором мы добавляем занчения в необходимые нам таблицы.

public class DaVaNiDataBase : DbContext

{

public DaVaNiDataBase() : base("name=DaVaNiConnection")

{

Database.SetInitializer(new DataBaseInitializer());

}

............

}

Для того что бы было удобно работать с таблицами БД нужно создать коллекции для каждой таблицы, такого вида:

public virtual DbSet<Customer> Customers { get; set; }

public virtual DbSet<Order> Orders { get; set; }

Для настройки таблиц при помощи «Fluent API» необходимо перегрузить метод «OnModelCreating», который принимает экземпляр «DbModelBuilder», часть реализации данного метода приведен ниже:

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Entity<Material.Material>()

.HasMany(e => e.MaterialParams)

.WithRequired(e => e.Material)

.HasForeignKey(e => e.MaterialId)

.WillCascadeOnDelete(false);

modelBuilder.Entity<MatType>()

.HasMany(e => e.ParamType)

.WithMany(e => e.MatType)

.Map(m => m.ToTable("MatParamType")

.MapLeftKey("MatTypeId")

.MapRightKey("ParamTypeId"));

. . . . . . .

}

Пример реализации классов модели приведен ниже:

[Table("Material")]

public class Material

{

public Material()

{

MaterialParams = new HashSet<MaterialParams>();

Deleted = false;

}

[DatabaseGenerated(DatabaseGeneratedOption.Identity)]

public int MaterialId { get; set; }

public MatTypeId MatTypeId { get; set; }

public virtual MatType MatType { get; set; }

public bool Deleted { get; set; }

public virtual ICollection<MaterialParams> MaterialParams { get; set; }

}

Данный класс предназначен для работы с материалами. Он хранит в себе идентификатор материала, ссылку на тип материала и его идентификатор, свойство для пометки материала удаленным, а также коллекцию параметров для данного материала.

* + 1. **Работа с библиотекой AutoMapper**

Для того, чтобы использовать «AutoMapper», нужно установить библиотеку из Nuget. Данная библиотека позволять автоматически преобразовать один объект в другой. Которая является очень удобной при необходимости связать классы Model с классами ViewModel. Ниже представлен пример создания мапинга двух классов:

class CustomerMapperProfile : Profile

{

protected override void Configure()

{

CreateMap<TelephoneNumber, AddEditCustomerContactsVM.Telephone>()

.ForMember(e => e.Id, r => r.MapFrom(e => e.Id))

.ForMember(e => e.CustomerId, r => r.MapFrom(e => e.CustomerId))

.ForMember(e => e.TelNumber, r => r.MapFrom(e => e.TelNumber));

CreateMap<AddEditCustomerContactsVM.Telephone, TelephoneNumber>()

.ForMember(e => e.Id, r => r.MapFrom(e => e.Id))

.ForMember(e => e.CustomerId, r => r.MapFrom(e => e.CustomerId))

.ForMember(e => e.TelNumber, r => r.MapFrom(e => e.TelNumber));

. . . . . . .

}

}

Для того что бы не создавать мапинги во всех классах приложения был создан класс унаследованный от класса «Profile» из библиотеки «AutoMapper», в котором необходимо реализовать перегрузку метода «Configure», в котором нужно создать все необходимые мапинги. В данном приложении было создано пять файлов профилей для разных частей программы. После того как код мапингов написан необходимо их создать. Для предотвращения многократного создания экземпляров профилей мы создаем их при запуске приложения, для этого мы в «App.xaml.cs» перегрузим метод «OnStartup»:

protected override void OnStartup(StartupEventArgs e)

{

var config = new MapperConfiguration(cfg =>

{

cfg.AddProfile(new ProductMapperProfile());

cfg.AddProfile(new EmployeeMapperProfile());

cfg.AddProfile(new CustomerMapperProfile());

cfg.AddProfile(new SupplierMapperProfile());

cfg.AddProfile(new MaterialMapperProfile());

});

Mapper = config.CreateMapper();

}

Данный метод будет срабатывать при запуске приложения, в середине данного метода мы создадим конфигурацию мапингов и запишем его в созданное открытое свойство «Mapper», к которому мы сможем обратится из любой части приложения.

* 1. **РАЗРАБОТКА ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА**
     1. **Главное окно**

Для разработки пользовательского интерфейса была использована технология WPF.

Основным элементом компоновки главного окна стал <DockPanel>, на котором визуально различаться три основных области.

Верхняя область – для отображения логотипа компании и в будущем появиться меню для добавления дополнительного функционала приложения.

<Border BorderBrush="#FF535353"

BorderThickness="0,0,0,1"

DockPanel.Dock="Top"

Padding="10,0,10,0">

<DockPanel LastChildFill="False">

<TextBlock DockPanel.Dock="Right"

FontSize="24"

Foreground="#FF828282"

FontFamily="ALGERIAN"

Text="DaVaNi" />

</DockPanel>

</Border>

Левая область – для элементов управления <Button/> реализующие переходи в основные разделы программы.

<Button x:Name="LeftPanel\_Product"

DockPanel.Dock="Top"

Content="Продукция"

Click="LeftPanel\_Product\_OnClick"/>

.........

Для того что бы получился такой задний фон левой панели было использовано следующий код:

<Border BorderBrush="#ff535353"

BorderThickness="1">

<Border.Background>

<LinearGradientBrush StartPoint="1,0.5" EndPoint="0,0.5">

<GradientStop Color="#FF545454" Offset="0"/>

<GradientStop Color="#CC202020" Offset="1"/>

</LinearGradientBrush>

</Border.Background>

</Border>

Правая область – которая является основной рабочей областью содержат в себе элемент <Frame/>, который используется для отображения всех страниц на которые будут совершаться переходы.

<Frame x:Name="MainAreaFrame"

Margin="5"/>

Для заднего фона основной области был использован линейный:

<LinearGradientBrush EndPoint="0.2,1.0" StartPoint="1.0,0.0">

<GradientStop Color="#FF202020" Offset="0"/>

<GradientStop Color="#FF545454" Offset="1"/>

</LinearGradientBrush>

* + 1. **Страницы с информацией**

Так же, как и в главном окне основным для всех страниц был использован элемент компоновки <DockPanel>, в котором легко и удобно размещать все нужные элементы.

Для отображения требуемой табличной информации был использован <DataGrid> с множеством настроек:

<DataGrid Margin="5,0,5,5"

VerticalScrollBarVisibility="Auto"

HorizontalScrollBarVisibility="Auto"

AutoGenerateColumns="False"

HorizontalContentAlignment="Center"

SelectionMode="Single"

SelectionUnit="FullRow"

ItemsSource="{Binding Items}"

SelectedItem="{Binding SelectedItem}">

.........

</DataGrid>

Непосредственно для отображения данных используются столбцы, которые можно использовать в стандартном виде для отображения строк информации,

<DataGridTextColumn Header="Поставщик"

IsReadOnly="True"

Width="50\*"

Binding="{Binding Path=Name}"/>

а также их можно преобразовать в нужный нам FrameworkElement, используя класс «DataGridTemplateColumn» и его свойства «CellTemplate», используя которые у нас появляется возможность преобразовать <DataTemplate> ячеек нашего столбца, на пример в кнопку, что мы и делаем в программе:

<DataGridTemplateColumn Header="Контакты"

IsReadOnly="True"

Width="25\*">

<DataGridTemplateColumn.CellTemplate>

<DataTemplate>

<Button Style="{x:Null}"

Content="Контакты"

Click="Contacts\_OnClick"/>

</DataTemplate>

</DataGridTemplateColumn.CellTemplate>

</DataGridTemplateColumn>

Для привязки информации был использован класс «Binding», который позволяет, используя MVVM сделать привязку элемента к свойству, которое хранит в себе нужное нам значение. В случае с <DataGrid> нам нужно привязать коллекцию определённого типа к свойству «ItemsSource», а также для того что бы получать выбранный элемент данного типа мы должны привязаться к свойству «SelectedItem»:

ItemsSource="{Binding Items}"

SelectedItem="{Binding SelectedItem}"

А в случае если нужно привязаться к обычному строковому значению нужно в свойстве «Path» указать имя свойства из ViewModel, которое отвечает за информацию в данной ячейке, но также нужно указать источник «Sourse» на класс из которого нужно подтягивать данные, в программе для удобства редактирования подключение к классу и указание источника данных это было вынесено в <Page.Resources> и <Page.DataContext> соответственно:

<Page.Resources>

<supplier:SupplierTableVM x:Key="Model"/>

</Page.Resources>

<Page.DataContext>

<Binding Source="{StaticResource Model}"/>

</Page.DataContext>

* + 1. **Окна добавления/изменения**

В программе используется три дополнительных окна, которые используются для добавления необходимой информации в соответствующие таблицы. Во всех трех окнах основным элементом компоновки является <Border>.

<Border Margin="10"

BorderThickness="1"

BorderBrush="#ff535353"

Background="{StaticResource changingWindowsGradient}">

</Border>

Фон связан с ресурсом, а так как он используется во всех окнах, он вынесен в <Application.Resources> для общего доступа для всех страниц и окон в приложении и представлен как <LinearGradientBrush>:

<LinearGradientBrush x:Key="changingWindowsGradient"

StartPoint="1,0.5" EndPoint="0,0.5">

<GradientStop Color="#FF545454" Offset="0"/>

<GradientStop Color="#CC202020" Offset="1"/>

</LinearGradientBrush>

Также общим для всех страниц и окон в приложения является стиль для кнопок, который отвечает за общие свойства в каждой кнопке:

<Style TargetType="Button">

<Setter Property="FontSize" Value="22"/>

<Setter Property="Margin" Value="5"/>

<Setter Property="FontFamily" Value="Monotype Corsiva" />

</Style>

Перейдем к рассмотрению используемых окон. Первое окно используется для добавления/изменения контактной информации клиентов и поставщиков. Что бы все поля и их названия были расположены на одном уровне друг к другу использовался элемент компоновки <DockPanel> в который был помещен еще один элемент компоновки <StackPanel> который располагает все поля и их названия горизонтально, и прикреплено к верхней грани окна за счет указания в <StackPanel> свойство «DockPanel.Dock="Top"» которое появилось за счет того что <StackPanel> обвернут элементом <DockPanel>.

<DockPanel LastChildFill="False">

<StackPanel DockPanel.Dock="Top"

Margin="5">

..........

</StackPanel>

</DockPanel>

Поля и их названия являются ничем иным как <TextBox> и <TextBlock> обвернутыми в элемент компоновки <Grid>, который дает возможность разместить эти элементы в два столбика у которых ширина будут заданы в процентном соотношении 30% для названия и 70% для поля:

<Grid Margin="5">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="30\*"/>

<ColumnDefinition Width="70\*"/>

</Grid.ColumnDefinitions>

<TextBlock Grid.Column="0"

Text="ФИО:"/>

<TextBox Grid.Column="1"

Text="{Binding Name }"/>

</Grid>

Но не все поля одинаковы хоть могут выглядеть таковыми, исключением является поле для ввода номера телефона. Данное поле является <ListBox> в котором было изменено представление его элементов как обычные <TextBox> для ввода текста, основным отличием является то что в случае надобности еще одного поля для телефона достаточно просто нажать на соответствующую кнопку:

<ListBox ItemsSource="{Binding TelephoneNumbers}"

HorizontalContentAlignment="Stretch">

<ListBox.ItemTemplate>

<DataTemplate>

<TextBox Text="{Binding TelNumber }"/>

</DataTemplate>

</ListBox.ItemTemplate>

</ListBox>

<Button x:Name="btnAddOneMoreTelephoneNumber"

Content="Добавить еще номер"

FontSize="10"

Foreground="AliceBlue"

Background="Transparent"

HorizontalAlignment="Right"

Click="btnAddOneMoreTelephoneNumber\_OnClick"/>

Кнопка добавляет еще одно поле к <ListBox> по той причине, что у кнопки есть обработчик события «Click» и метод для его обработки.

Второе окно так же используется для добавления/изменения, но в этот раз уже сотрудника, и это окно имеет несколько отличий. Первым и самым большим отличием является возможность добавления любого количества электронных копий документов, которые необходимы для приема на работу. Для этого имеется кнопка с соответствующим событием «Click» и метод для его обработки.

<TextBlock Grid.Column="0"

Text="Добавить документы:"/>

<Button x:Name="btnAddDocumentPhoto"

Grid.Column="1"

Margin="0"

Content="Добавить"

FontSize="20"

Click="BtnAddDocumentPhoto\_OnClick"/>

Также есть возможность пред просмотра и удаления добавленных документов при помощи элемента <ListView> в котором был изменен стиль группировки элементов при помощи свойств <GroupStyle> и <ItemsPanelTemplate> на <WrapPanel> для более удобной компоновки документов. Также был изменен шаблон представления элементов при помощи <DataTemplate> на более сложный шаблон в виде элементов <Image> и <Button> которые в свою очередь обвернуты в элемент компоновки <DockPanel>. У элемента <Image> к свойству «Source» произведена привязка к свойству из ViewModel, а также к данной привязке добавлено свойство «Converter» с подключённым к нему ресурсом, который является классом, который преобразовывает изображение в массив бит и наоборот.

<ListView Background="Transparent"

Height="200"

ItemsSource="{Binding Documents}"

SelectedItem="{Binding SelectedDocument}">

<ListView.GroupStyle>

<GroupStyle>

<GroupStyle.Panel>

<ItemsPanelTemplate>

<WrapPanel Width="370"

HorizontalAlignment="Left"/>

</ItemsPanelTemplate>

</GroupStyle.Panel>

</GroupStyle>

</ListView.GroupStyle>

<ListView.ItemTemplate>

<DataTemplate>

<DockPanel Margin="3">

<Image Source="{Binding Data,

Converter={StaticResource Converter}}"

MaxWidth="100"

MaxHeight="100"

Stretch="Uniform"/>

<Button Content="Delete"

DockPanel.Dock="Bottom"

Click="BtnDeletePhoto\_OnClick"/>

</DockPanel>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

Последним отличием является элемент <DatePicker/> который позволяет удобно вводить/ выводить дату.

Третье окно внешне не отличается от первого практически ничем, только оно предназначено для добавления материалов, но вот в кодовой части самым явным отличием является то что мы динамически создаем нужные нам поля в зависимости от того какой материал выбран в выпадающем списке первого поля:

<ComboBox Style="{StaticResource ComboBoxStyle}"

ItemsSource="{Binding MaterialNames}"

SelectedItem="{Binding SelectedMaterial}">

<ComboBox.ItemTemplate>

<DataTemplate>

<TextBlock Text="{Binding Name}"/>

</DataTemplate>

</ComboBox.ItemTemplate>

</ComboBox>

Создание динамических полей производится в <ListView> в котором источником значений является коллекция класса параметров, а также шаблон элементов представлен с помощью ресурса – специального класса в котором идет определение какой элемент нужно создавать:

<ListView ItemsSource="{Binding ParamValues}"

BorderThickness="0"

ItemTemplateSelector="{StaticResource TemplateSelector}"

Background="Transparent"

HorizontalContentAlignment="Stretch"/>

В данном окне используется три шаблона элементов: текстовое поле, выпадающий список и радио кнопка, ниже приведен пример используемого шаблона для выпадающего списка:

<DataTemplate x:Key="DropdownTemplate"

DataType="mat:DropdownParamValue">

<StackPanel>

<Grid Margin="5">

<Grid.ColumnDefinitions>

<ColumnDefinition Width="30\*" />

<ColumnDefinition Width="70\*" />

</Grid.ColumnDefinitions>

<TextBlock Style="{StaticResource TextBlockStyle}"

Grid.Column="0"

Text="{Binding ParametrName}" />

<ComboBox Style="{StaticResource ComboBoxStyle}"

Grid.Column="1"

ItemsSource="{Binding Items}"

SelectedItem="{Binding ParametrValue}"/>

</Grid>

</StackPanel>

</DataTemplate>

В зависимости то того какой надо создать элемент вызывается <DataTemplate> с соответствующим ключом «x:Key».

1. **РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**
   1. **Главное Окно**

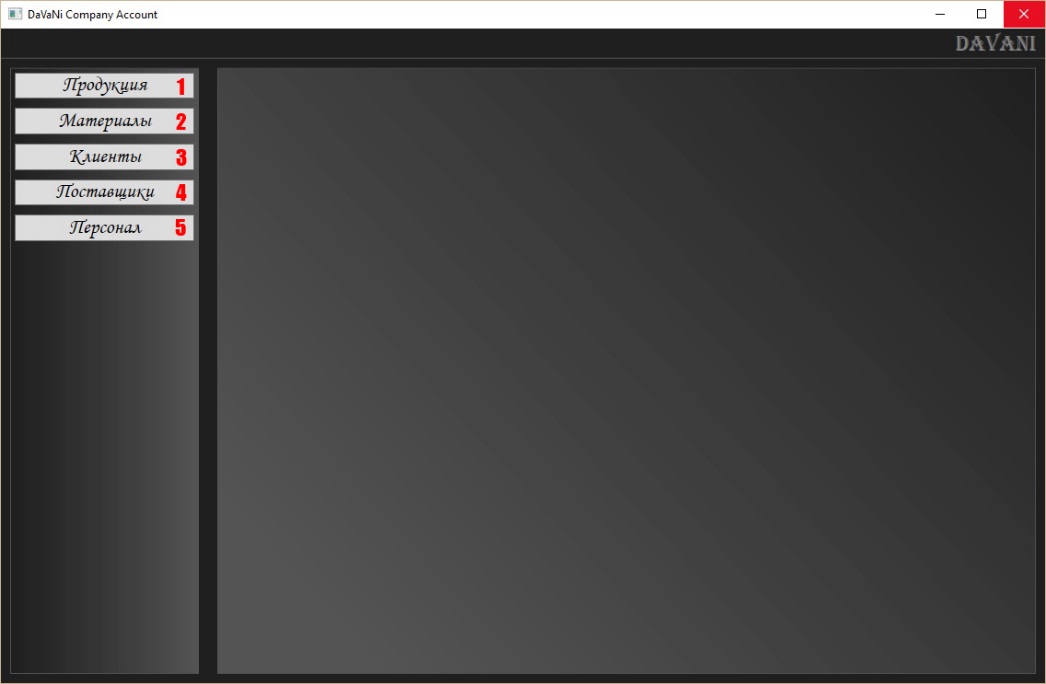


Рис.1 – Главное окно приложения.

1 – кнопка для перехода на страницу продукции, рис.2;

2 – кнопка для перехода на страницу материалов, рис.6;

3 – кнопка для перехода на страницу клиентов, рис.9;

4 – кнопка для перехода на страницу поставщиков, рис.14;

1. – кнопка для перехода на страницу персонала, рис.17;
   1. **Страница продукции**



Рис.2 – Страница продукции.

1 – кнопка для перехода на страницу с таблицей продукции из овчины, рис.3;

2 – кнопка для перехода на страницу с таблицей продукции из телятины, рис.3;

3 – кнопка для перехода на страницу с таблицей продукции из дестроя, рис.3;

4 – кнопка для перехода на страницу с таблицей продукции из текстиля, рис.3;

5 – кнопка для перехода на страницу с таблицей отображения всей имеющейся продукции, рис.3;

* + 1. **Таблица продукции**

На рис.3 изображен шаблон таблицы для отображения продукции.

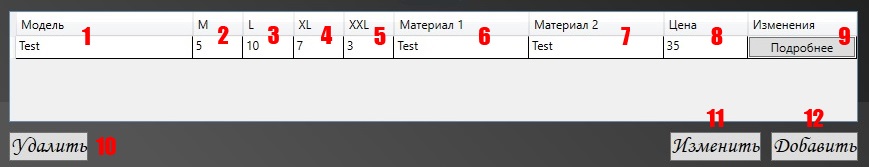


Рис.3 – Страница продукции.

1 – столбец для отображения названия модели;

2 – столбец для отображения количества размера М данной модели;

3 – столбец для отображения количества размера L данной модели;

4 – столбец для отображения количества размера XL данной модели;

5 – столбец для отображения количества размера XXL данной модели;

6,7 – столбцы для отображения названий материалов;

8 – столбец для отображения цены на единицу данной модели;

9 – кнопка для перехода на страницу с историей изменений данных выбранного изделия, рис.4;

10 – кнопка для удаления выбранного изделия;

11 – кнопка для открытия окна изменения выбранного изделия, рис.5;

1. – кнопка для открытия окна добавления нового изделия, рис.5;
   * 1. **Таблица изменений, данных о изделиях**

В верхней части страницы отображается название модели изделия, к которому относится история данных изменений.

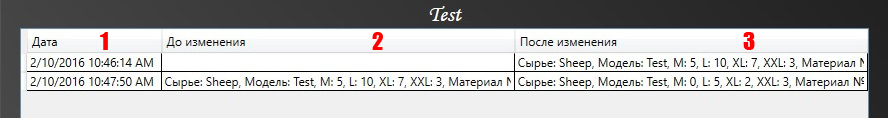


Рис.4 – Таблицы изменений изделия.

1 – столбец для отображения даты и времени изменений;

2 – столбец для отображения информации о том какими были данные до момента изменения;

3 – столбец для отображения информации о том какими стали данные после изменения;

* + 1. **Окно изменения/добавления изделий**



Рис.5 – Окно изменения/добавления изделий.

1 – строка ввода названия модели;

2 – строка ввода количества размера М данной модели;

3 – строка ввода количества размера L данной модели;

4 – строка ввода количества размера XL данной модели;

5 – строка ввода количества размера XXL данной модели;

6,7 – строки ввода названий материалов;

8 – столбец для отображения цены на единицу данной модели;

9 – кнопка отмены изменения/добавления изделия;

1. – кнопка для подтверждения изменения/добавления изделия;
   1. **Страница материалов**



Рис.6 – Страница материалов.

1 – кнопка для перехода на страницу с таблицей материалов – текстиль, рис.7;

2 – кнопка для перехода на страницу с таблицей материалов – кожа, рис.7;

3 – кнопка для перехода на страницу с таблицей материалов – мех, рис.7;

4 – кнопка для перехода на страницу с таблицей материалов – подкладка, рис.7;

5 – кнопка для перехода на страницу с таблицей материалов – латекс, рис.7;

6 – кнопка для перехода на страницу с таблицей материалов – поролон, рис.7;

7 – кнопка для перехода на страницу с таблицей материалов – синтепон, рис.7;

8 – кнопка для перехода на страницу с таблицей материалов – козырьки, рис.7;

9 – кнопка для перехода на страницу с таблицей материалов – нитки, рис.7;

* + 1. **Таблица материалов**

На рис.7 изображен шаблон таблицы для отображения материалов.

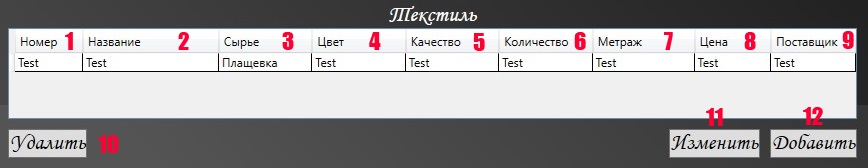


Рис.7 – Таблица материалов.

1 – столбец для отображения номера материала;

2 – столбец для отображения названия материала;

3 – столбец для отображения сырья материала;

4 – столбец для отображения цвета материала;

5 – столбец для отображения качества материала;

6 – столбец для отображения количества материала;

7 – столбец для отображения метража материала;

8 – столбец для отображения цены материала;

9 – столбец для отображения поставщика данного материала;

10 – кнопка для удаления выбранного материала;

11 – кнопка для открытия окна изменения выбранного материала, рис.8;

1. – кнопка для открытия окна добавления нового материала, рис.8;
   * 1. **Окно изменения/добавления материала**

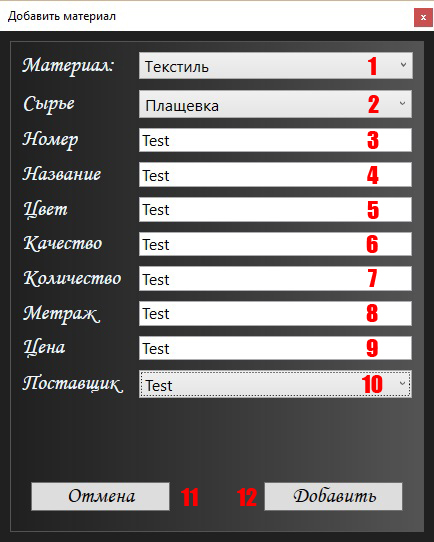


Рис.8 – Окно изменения/добавления материала.

1 – выпадающий список, для выбора нужного типа материала;

2 – выпадающий список, для выбора сырья;

3 – строка ввода номера материала;

4 – строка ввода названия материала;

5 – строка ввода цвета материала;

6 – строка ввода качества материала;

7 – строка ввода количества материала;

8 – строка ввода метража материала;

9 – строка ввода цены материала;

10 – выпадающий список для выбора поставщика из таблицы поставщиков;

11 – кнопка отмены изменения/добавления материала;

1. – кнопка для подтверждения изменения/добавления материала;
   1. **Страница клиентов**

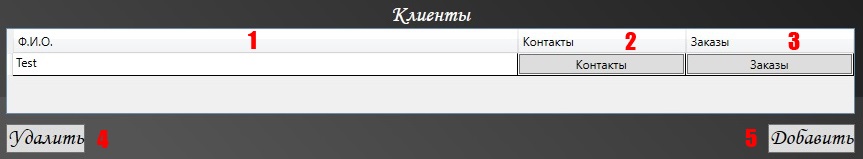


Рис.9 – Таблица клиентов.

1 – столбец для отображения фамилии, имени и отчества клиента;

2 – кнопка для открытия окна с контактными данными клиента, рис.10;

3 – кнопка для перехода на страницу с таблицей заказов выбранного клиента, рис.11;

4 – кнопка для удаления выбранного клиента со всеми его контактами и заказами;

5 – кнопка для открытия окна добавления контактных данных нового клиента, рис.13;

* + 1. **Окно с контактными данными**

Данное окно используется для отображения контактной информации клиентов и поставщиков, информация в данном окне только для ознакомления, в случае потребности изменить некоторые или все данный нужно нажать на кнопку №7 – «Изменить».

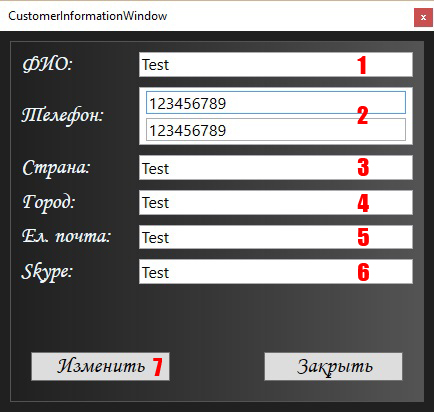


Рис.10 – Окно просмотра контактной информации

1 – строка отображения фамилии, имени и отчества человека;

2 – строки отображения номеров телефона человека;

3 – строка отображения страны проживания человека;

4 – строка отображения города проживания человека;

5 – строка отображения электронной почты человека;

6 – строка отображения имени для скайпа;

7 – кнопка для открытия окна изменения контактных данных клиента, рис.13;

* + 1. **Таблица всех заказов**

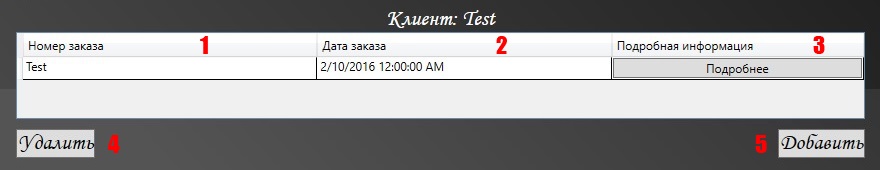


Рис.11 – Таблица всех заказов.

1 – столбец для отображения номера заказа;

2 – столбец для отображения даты заказа;

3 – кнопка для открытия страницы с подробной информацией о выбранном заказе, рис.12;

4 – кнопка для удаления выбранного заказа;

5 – кнопка для открытия окна добавления нового заказа для данного клиента, рис.12;

* + 1. **Таблица изменения/добавления заказа**



Рис.12 – Таблица изменения/добавления заказа.

1 – столбец для отображения номера заказа;

2 – столбец для отображения модели изделия;

3 – столбец для отображения количество изделий данной модели;

4 – столбец для отображения цена за единицу изделия данной модели;

5 – столбец для отображения цени за все изделия данной модели;

6 – столбец для отображения даты заказа;

7 – кнопка для удаления выбранной модели изделия из заказа;

1. – кнопка для сохранения всех изменений произведенных в заказе;
   * 1. **Окно изменения/добавления контактной информации**

Данное окно используется для изменения/добавления контактной информации клиентов и поставщиков.

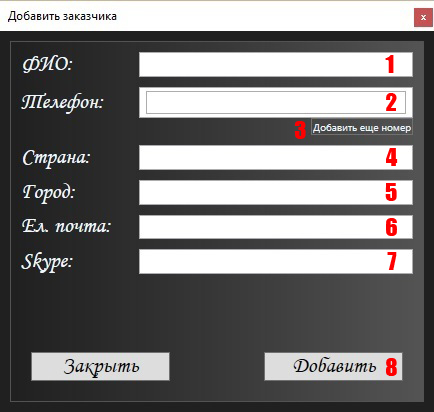


Рис.13 – Окно изменения/добавления контактной информации.

1 – строка ввода фамилии, имени и отчества человека;

2 – строки ввода номеров телефона человека;

3 – кнопка добавления дополнительного номера телефона;

4 – строка ввода страны проживания человека;

5 – строка ввода города проживания человека;

6 – строка ввода электронной почты человека;

7 – строка ввода имени для скайпа;

8 – кнопка для подтверждения изменения/добавления контактной информации;

* 1. **Страница поставщиков**

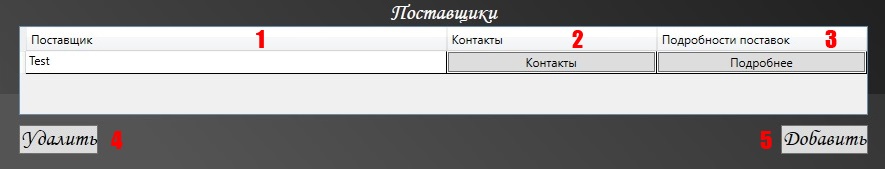


Рис.14 – Таблица поставщиков.

1 – столбец для отображения название (Ф.И.О.) поставщика;

2 – кнопка для открытия окна с контактными данными поставщика, рис.10;

3 – кнопка для перехода на страницу с таблицей информации поставок от выбранного поставщика, рис.15;

4 – кнопка для удаления выбранного поставщика со всеми его контактами и информацией о поставках;

5 – кнопка для открытия окна добавления контактных данных нового поставщика, рис.13;

* + 1. **Страница поставок**

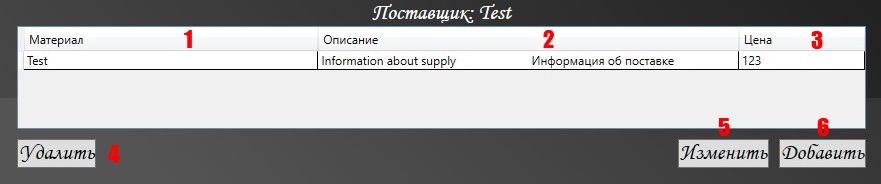


Рис.15 – Таблица поставок.

1 – столбец для отображения название материала поставки;

2 – столбец для отображения описания поставки;

3 – столбец для отображения цены на материал поставки;

4 – кнопка для удаления выбранной поставки;

5 – кнопка для открытия окна изменения информации о поставке, рис.16;

6 – кнопка для открытия окна добавления информации о новой поставке, рис.16;

* + 1. **Окно изменения/добавления информации о поставке**

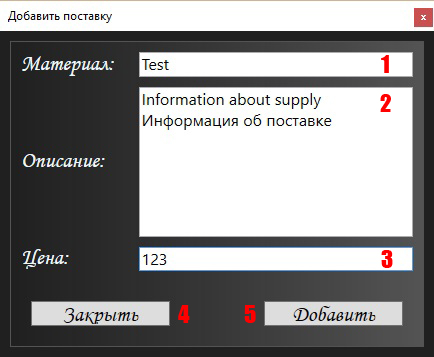


Рис.16 – Окно изменения/добавления информации о поставке.

1 – строка ввода название материала поставки;

2 – строка ввода описания поставки;

3 – строка ввода цены на материал поставки;

4 – кнопка отмены изменения/добавления информации о поставке;

5 – кнопка для подтверждения изменения/добавления информации о поставке;

* 1. **Страница персонала**

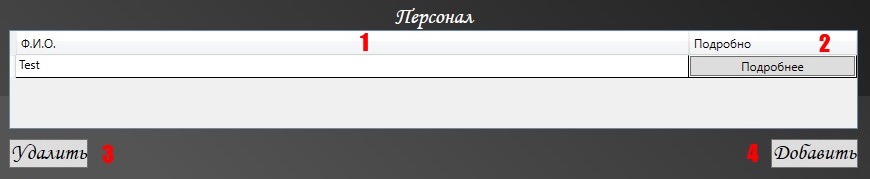


Рис.17 – Таблица персонала.

1 – столбец для отображения Ф.И.О. сотрудника;

2 – кнопка для открытия окна с данными сотрудника, рис.18;

3 – кнопка для удаления выбранного сотрудника со всеми его контактами;

4 – кнопка для открытия окна добавления контактных данных нового сотрудника, рис.19;

* + 1. **Окно с контактными данными сотрудника**

Информация в данном окне только для ознакомления, в случае потребности изменить некоторые или все данный нужно нажать на кнопку №8 – «Изменить»

1 – строка отображения фамилии, имени и отчества сотрудника;

2 – строки отображения номеров телефона сотрудника;

3 – строка отображения даты приема на работу сотрудника;

4 – строка отображения должности сотрудника;

5 – строка отображения заработной платы сотрудника;

6 – выпадающий список для выбора документа для просмотра;

7 – область для просмотра изображения документа;

1. – кнопка для открытия окна изменения данных сотрудника, рис.19;

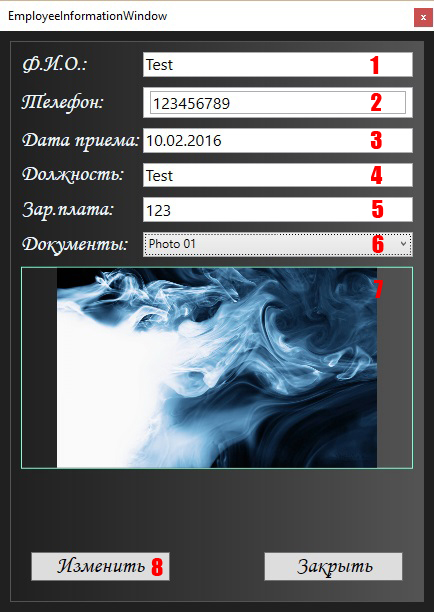


Рис.18 – Окно просмотра контактной информации сотрудника

* + 1. **Окно изменения/добавления данных о сотруднике**



Рис.18 – Окно изменения/добавления данных о сотруднике

1 – строка ввода фамилии, имени и отчества сотрудника;

2 – строки ввода номеров телефона сотрудника;

3 – кнопка добавления дополнительного номера телефона;

4 – ввод даты приема на работу сотрудника;

5 – строка ввода должности сотрудника;

6 – строка ввода заработной платы человека;

7 – область пред просмотра добавленных документов;

8 – строка ввода заработной платы человека;

9 – кнопка для добавления документов сотрудника;

10 – кнопка для подтверждения изменения/добавления данных о сотруднике.