Министерство образования Рязанской области

Областное государственное бюджетное профессиональное образовательное учреждение

«Касимовский нефтегазовый колледж»

**Методические рекомендации**

**по выполнению учебной практики**

**Профессионального модуля 05**

**«Разработка и проектирование информационных систем»**

Разработал:

Преподаватель специальных дисциплин

Борковской И.Б.

При поддержке студента гр.21И-1

Пышкина Артема

Касимов 2024

**Содержание**

[**Анализ предметной области** 3](#_Toc162808335)

[**Функциональная модель информационной системы** 4](#_Toc162808336)

[**Моделирование процессов информационной системы** 9](#_Toc162808337)

[**Логическая модели базы данных** 14](#_Toc162808338)

[**Физическая модель данных** 15](#_Toc162808339)

[**Создание Базы данных** 16](#_Toc162808340)

[**Реализация информационной системы** 25](#_Toc162808341)

[**Функциональное тестирование** 74](#_Toc162808342)

# **Анализ предметной области**

Предметная область сильно влияет на все аспекты проекта: требования к системе, взаимодействие с пользователем, модель хранения данных, реализацию и т.д.

Анализ предметной области, позволяет выделить ее сущности, определить первоначальные требования к функциональности и определить границы проекта. Модель предметной области должна быть документирована, храниться и поддерживаться в актуальном состоянии до этапа реализации. Для документирования могут быть использованы различные средства.

В данной главе необходимы пункты:

* Цель создания ИС
* Назначение ИС
* Автоматизируемые функции

**Пример:**

Целью создания информационной системы является автоматизация учета информации о приеме пациентов.

Назначение АИС:

- Хранение информации о врачах клиники;

- Хранение информации о пациентах;

- Хранение информации о приемах, в том числе диагнозах и лечении, а также стоимости лечения;

- Обновление и добавление информации;

- Выдача итоговой информации в виде отчетов.

Перечень автоматизируемых функций:

1. оформление личной карточки пациента
2. хранение личной карточки пациента историй всех болезней поставленных диагнозов результатов проведения исследований анализов
3. направление пациента к врачу на проведение исследования сдачу анализов
4. централизованное хранение всех данных о пациенте

# **Функциональная модель информационной системы**

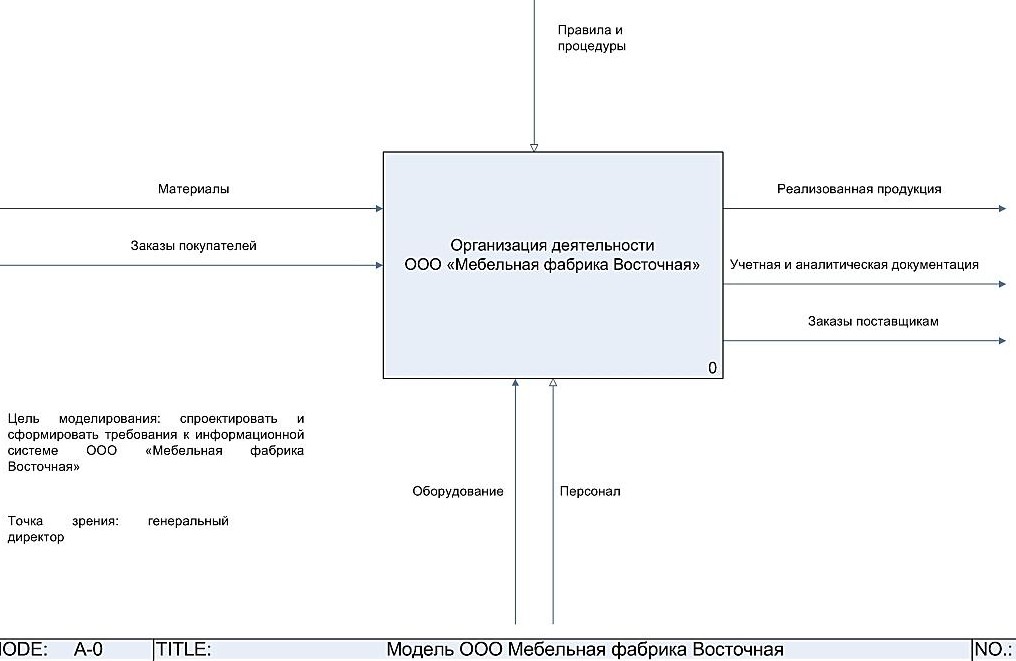
IDEF0 – нотация графического моделирования, используемая для создания функциональной модели, отображающей структуру и функции системы, а также потоки информации и материальных объектов, связывающих эти функции.

К ее особенностям можно отнести:

1. использование контекстной диаграммы;
2. поддержку декомпозиции;
3. доминирование;
4. выделение четырех типов стрелок.

Контекстная диаграмма – это верхняя диаграмма, на которой объект моделирования представлен единственным блоком с граничными стрелками. Эта диаграмма называется A-0. Стрелки на этой диаграмме отображают связи объекта моделирования с окружающей средой. Диаграмма A-0 устанавливает область моделирования и ее границу.

Контекстная диаграмма рассматриваемой предметной области «Мебель- ная фабрика Восточная» представлена на рис. 1.1:



*Рис. 1.1.* Контекстная диаграмма предприятия «Мебельная фабрика Восточная»

Поддержка декомпозиции – это принцип проектирования, согласно которому нотация IDEF0 детализирует последовательную декомпозицию процесса до требуемого уровня детализации. Дочерняя диаграмма, создаваемая при декомпозиции, охватывает ту же область, что и родительский процесс, но описывает ее более подробно.

Суть принципа доминирования заключается в том, что блоки модели IDEF0 на неконтекстной диаграмме должны располагаться по диагонали – от левого верхнего угла диаграммы до правого нижнего в порядке присвоенных номеров. Блоки на диаграмме, расположенные вверху слева, «доминируют» над блоками, расположенными внизу справа. Под «доминированием» понимается влияние, которое блок оказывает на другие блоки диаграммы.

Используются следующие типы стрелок: «вход», «выход», «механизм», «управление». «Входы» преобразуются или расходуются процессом, чтобы создать то, что появится на выходе. «Управления» определяют условия, необходимые процессу, чтобы получить правильный результат на выход. «Выходы» – данные или материальные объекты, полученные в ходе выполнения процесса. «Механизмы» идентифицируют средства, поддерживающие выполнение процесса.

Элементы графической нотации IDEF0 представлены в табл. 1.1.

*Таблица 1.1*

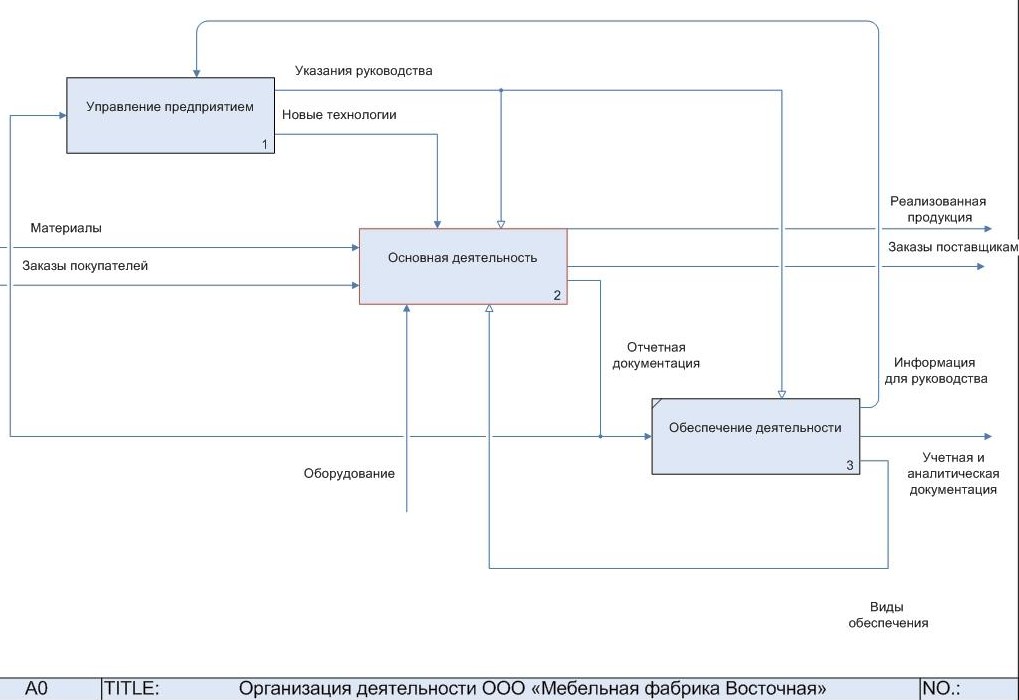
**Назначение графических символов, используемых в нотации IDEF0**

|  |  |  |  |
| --- | --- | --- | --- |
| **Номер**  **п/п** | **Название**  **элемента** | **Графическое**  **обозначение** | **Описание**  **элемента** |
| 1 | Процесс |  | Обозначается прямоугольным блоком. Внутри каждого блока помещаются его имя и номер. Имя должно быть актив-ным глаголом, глагольным оборотом или отглагольным существительным. Номер блока размещается в правом нижнем углу. Номера блоков исполь- зуются для идентификации на диаграм-  ме и в соответствующем тексте |
| 2 | Стрелка |  | Стрелки обозначают входящие и исхо- дящие из процесса объекты (данные).  Каждая сторона функционального блока имеет стандартное значение с точки зре- ния связи «блок – стрелка». В свою оче- редь, сторона блока, к которой присо- единена стрелка, однозначно определяет ее роль. Стрелки, входящие в левую сто- рону блока, – «входы». Стрелки, входя- щие в блок сверху, – «управления». Стрелки, покидающие процесс справа, –  «выходы», т. е. данные или материаль- ные объекты, произведенные процессом. Стрелки, подключенные к нижней сто-  роне блока, представляют механизмы |

*Окончание табл. 1.1*

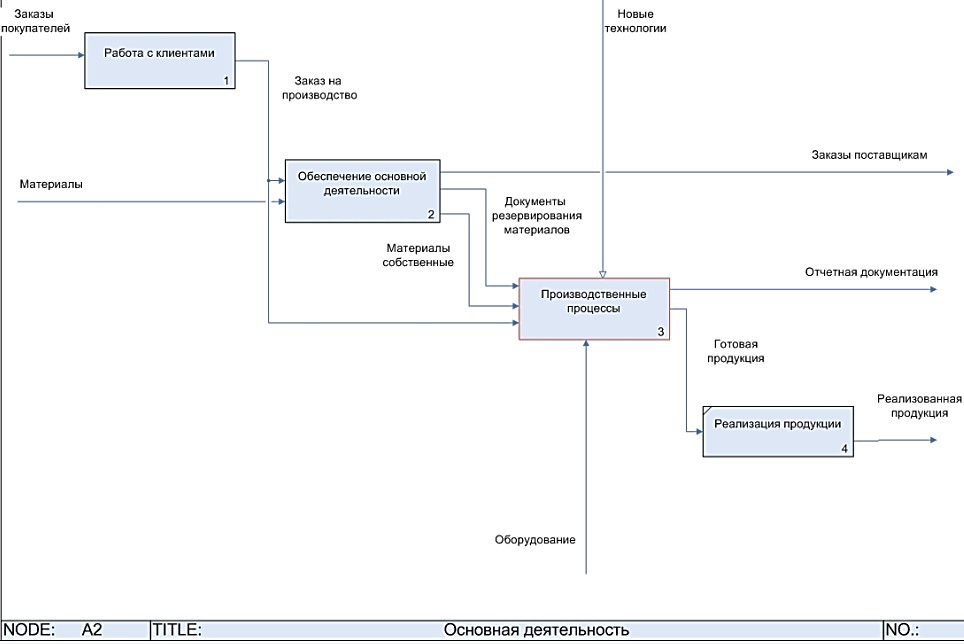
|  |  |  |  |
| --- | --- | --- | --- |
| **Номер**  **п/п** | **Название**  **элемента** | **Графическое**  **обозначение** | **Описание**  **элемента** |
| 3 | Туннели- рованная стрелка |  | Туннелированные стрелки означают, что данные, передаваемые с помощью этих стрелок, не рассматриваются на роди- тельской диаграмме и/или на дочерней диаграмме. Стрелка, помещенная в тун- нель там, где она присоединяется к бло- ку, означает, что данные, выраженные этой стрелкой, не обязательны на сле- дующем уровне декомпозиции. Стрелка, помещаемая в туннель на свободном конце, означает, что выраженные ею данные отсутствуют на родительской  диаграмме |
| 4 | Внешняя ссылка |  | Элемент обозначает место, сущность или субъект, которые находятся за границами моделируемой системы. Внешние ссыл- ки используются для обозначения источ- ника или приемника стрелки вне модели. На диаграммах внешняя ссылка изобра- жается в виде квадрата, рядом с которым  показано наименование внешней ссылки |
| 5 | Сноска |  | Выносной элемент, предназначенный для комментариев |

Согласно представленному выше описанию предметной области, можно выделить основные процессы, управляющие процессы, а также процессы, необходимые для обеспечения основной деятельности предприятия. В примере сделан акцент на основные процессы мебельной фабрики, среди которых выделяются работа с клиентами, обеспечение основной деятельности, производственные процессы, реализация продукции. Отправной точкой процесса можно считать документ «Заказы покупателей», а также необходимые для выполнения заказов ресурсы (материалы) и наличие персонала. Для разработки программы планирования необходимо определить затраты на производство и сформировать документы о резервировании материалов. На программу планирования производства также влияют отчетные показатели по изготовлению продукции. На основе разработанной программы планирования производства утверждается задание на производство.

Декомпозиция контекстной диаграммы до третьего уровня детализации бизнес-процессов представлена на рис. 1.2, 1.3, 1.4.

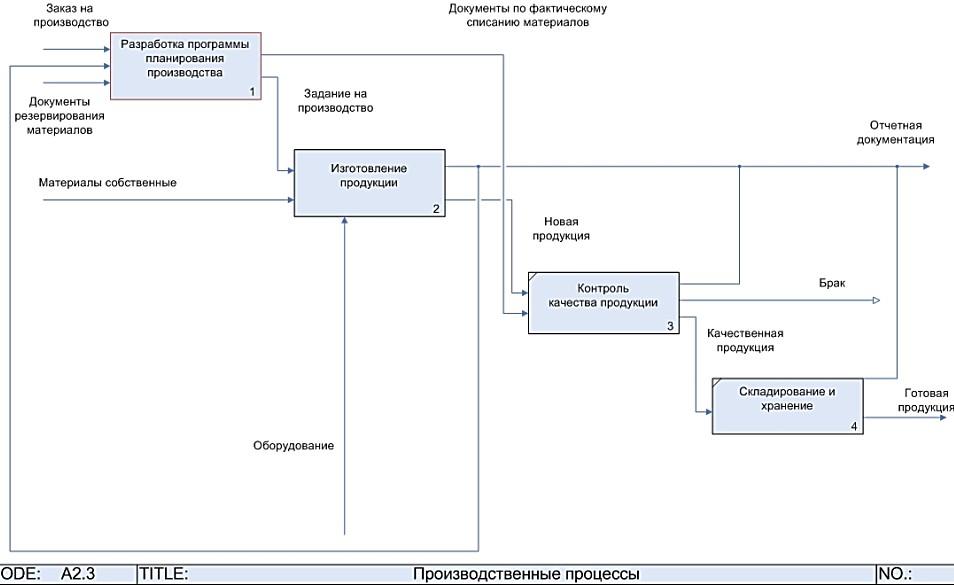
*Рис. 1.2.* Диаграмма декомпозиции бизнес-процессов предприятия

«Мебельная фабрика Восточная». Уровень A0



*Рис. 1.3.* Диаграмма декомпозиции бизнес-процессов предприятия

«Мебельная фабрика Восточная». Уровень A2



*Рис. 1.4.* Диаграмма декомпозиции бизнес-процессов предприятия

«Мебельная фабрика Восточная». Уровень A2.3

Таким образом, нотация IDEF0 предполагает построение иерархической системы диаграмм – единичных описаний фрагментов системы.

# **Моделирование процессов информационной системы**

**Моделирование процессов ИС выполняется с помощью UML-диаграмм.**

**UML**, или **Unified Modeling Language**, — это унифицированный язык моделирования. Его используют, чтобы создавать диаграммы и схемы для визуализации процессов и явлений.

Слово «унифицированный» означает, что схемы на UML будут понятны всем, кто знаком с ним. То есть у языка есть определённые правила, по которым применяются все стрелочки, кружки и квадраты.

Диаграммы, которые обязательны в отчете:

**Диаграмма вариантов использования (Диаграмма прецедентов).**

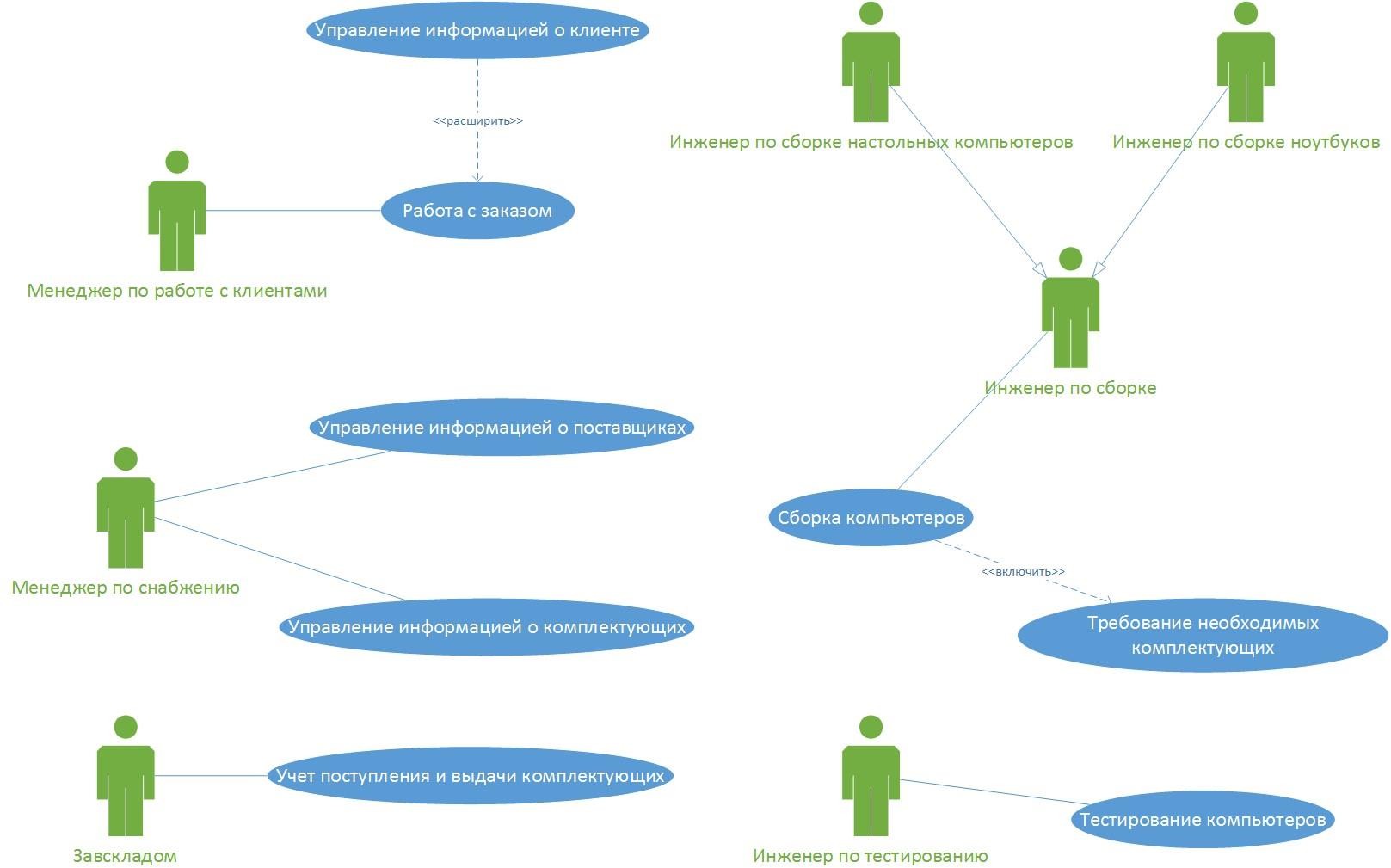
Данная диаграмма предназначена для построения модели, которая будет отражать функционирование системы в окружающей среде.

При построении диаграммы используют следующие элементы:

* действующее лицо (актер) – обозначает роль пользователя, взаимодействующего с какой-либо определенной сущностью. Действующим лицом может быть не только человек, но и объекты внешнего мира;
* прецеденты – отображают действия, выполняемые системой, которые приводят к результатам, наблюдаемым действующими лицами.

Между прецедентами существуют связи:

* обобщение – показывает общность ролей;
* включение – показывает связь нескольких прецедентов, где базовый использует функциональное поведение связанных с ним вариантов использования;
* расширение – показывает взаимосвязь базового прецедента и прецедентов, которые являются специальными случаями.



**Диаграмма классов.**

Диаграмма классов представляет детальную информацию о структуре модели системы, т.е. о внутреннем устройстве системы. Указывается внутренняя структура системы и типы отношений между отдельными подсистемами и объектами.

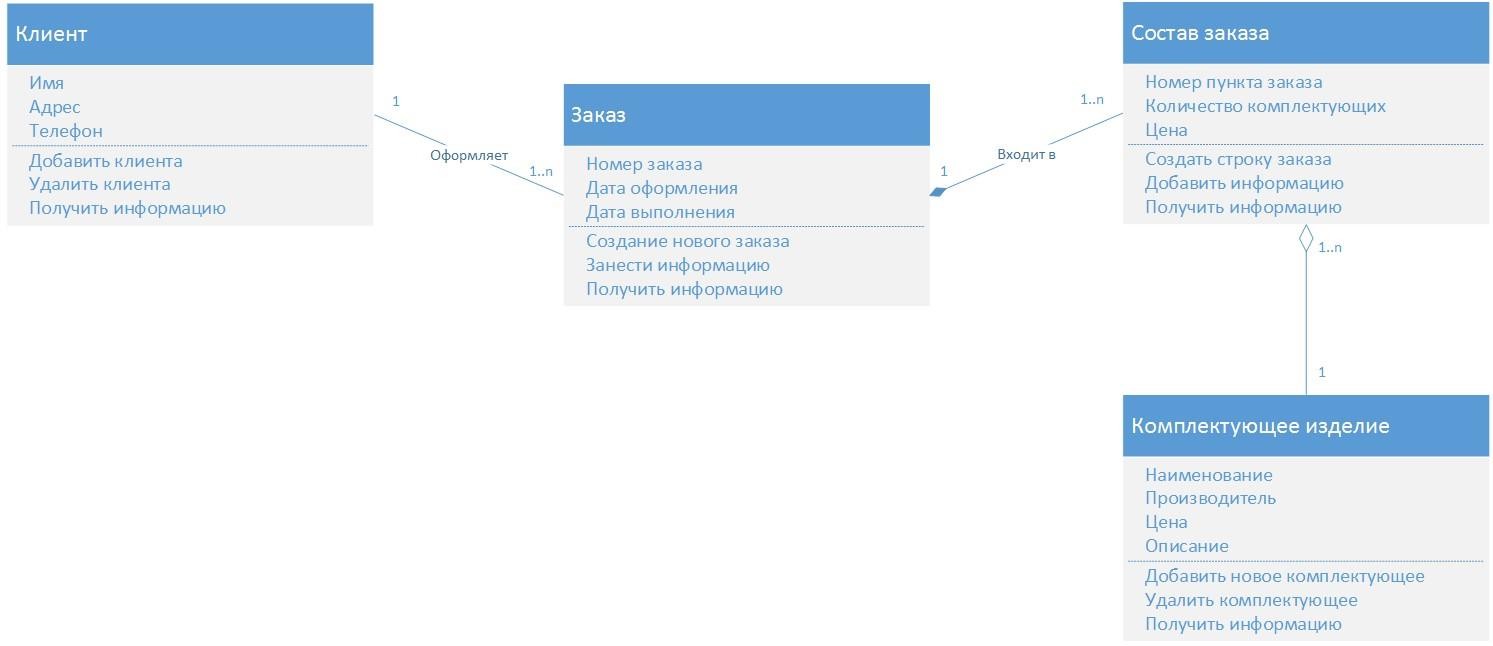
Класс в UML – это множество объектов, которые имеют одинаковую структуру и взаимосвязь с объектами других классов.

Диаграммы классов позволяют структурировать требования к элементам, данным, функциональности и интерфейсам проектируемой системы.

Элементы моделей классов отображаются в элементах базы данных и приложений:

* классы – в таблицах;
* атрибуты – в столбцах;
* типы – в типах данных СУБД;
* ассоциации – в связях между таблицами;
* приложения – в классах с определенными методами и атрибутами.

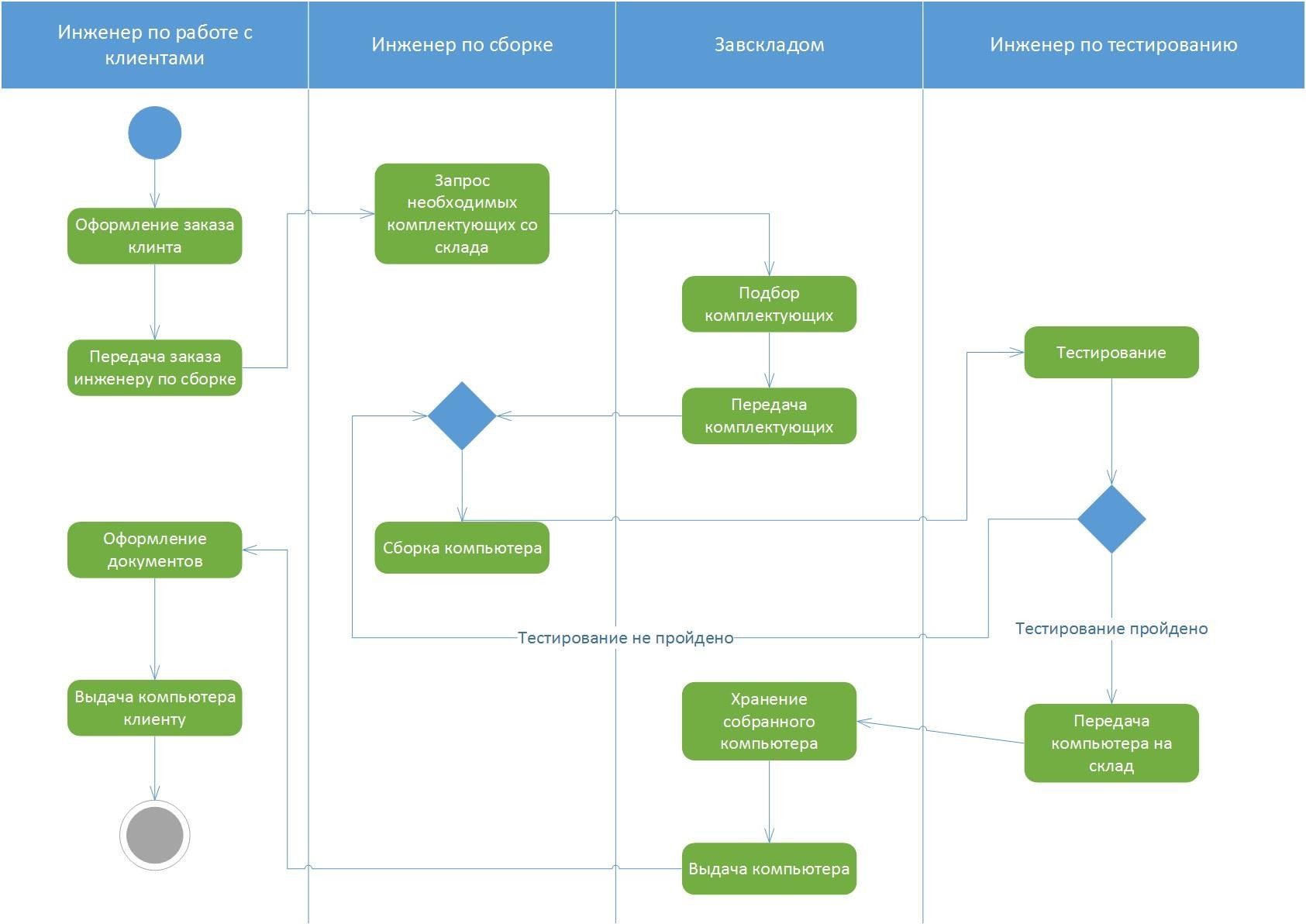
В представлении модели для каждого выделенного класса формируется таблица, включающая столбец, соответствующий атрибутам класса.



**Диаграмма деятельности.**

Диаграмма, отражающая разложение на составные части какой-либо деятельности. Она представляет выполнение отдельных действий и вложенных видов деятельности, соединяемые между собой потоками от выходов одного узла к входам другого, с указанием исполнителей.

Диаграмма деятельности имеет свойство корректировки по выявлению новых подробностей в описании бизнес-процесса объекта автоматизации на этапах анализа и проектирования.



**Диаграмма последовательности.**

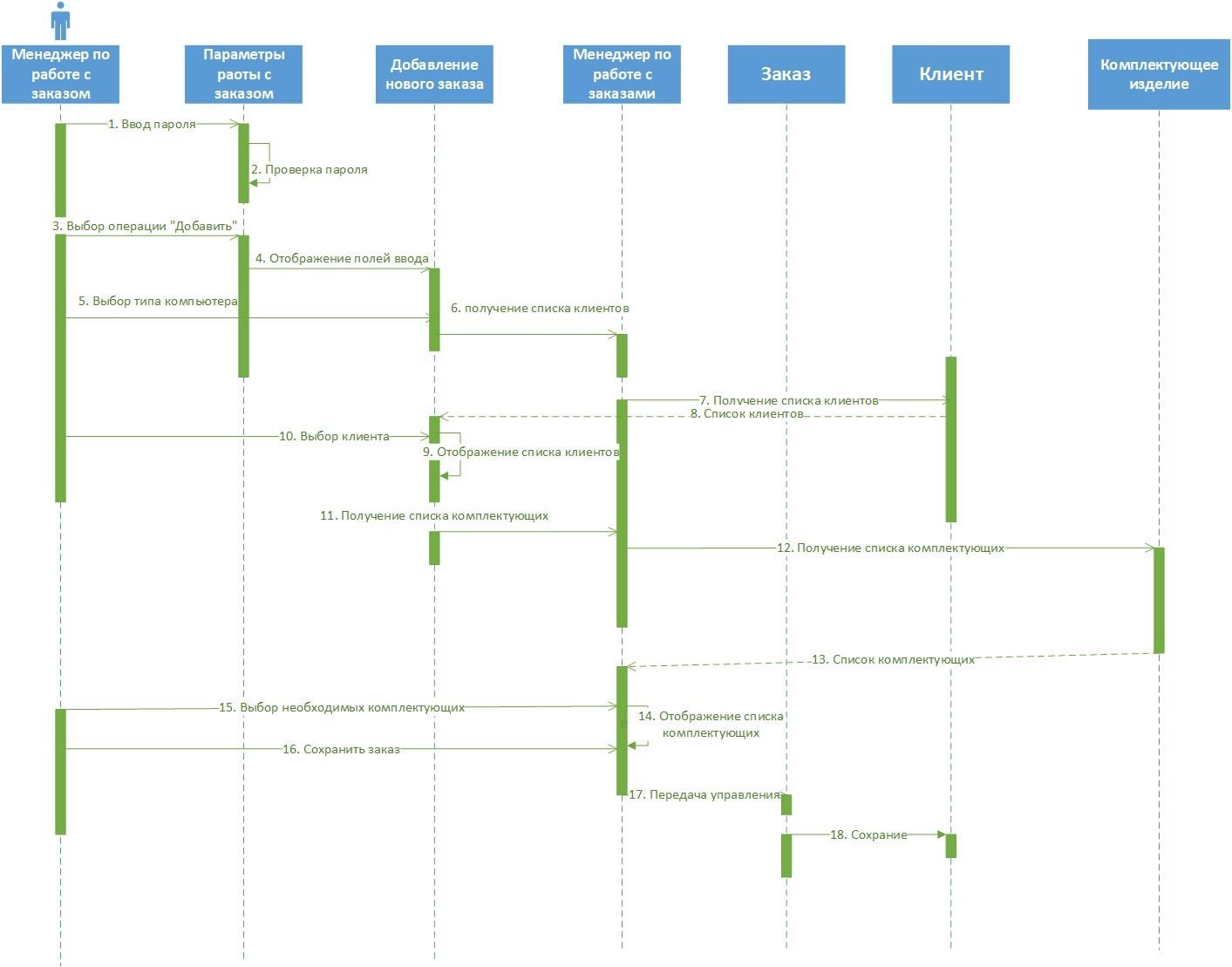
Диаграмма последовательности отражает упорядоченные по времени проявления взаимодействия объектов.

На диаграмме слева направо отображаются основные элементы, такие

как:

* объекты;
* линии жизни – моделируют течение времени при выполнении действий объектом;
* стрелки – определяют действия, выполняемые объектом.

При построении схемы разрабатываются подробные описания действий специалистов по внедрению информационной системы, необходимых для обеспечения ее функциональности.

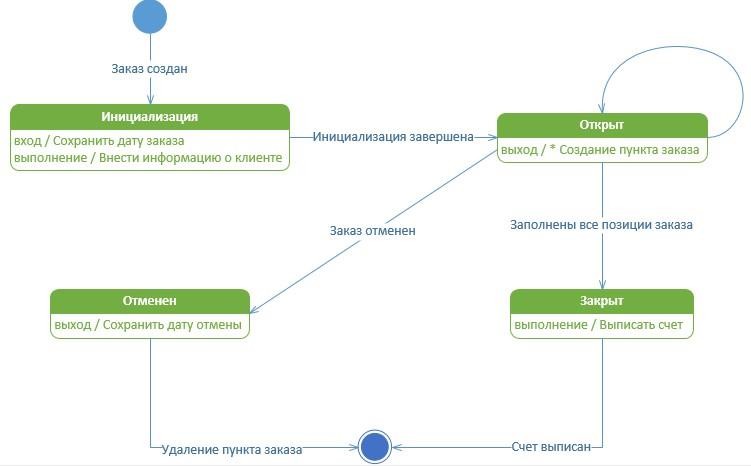


**Конечный автомат UML (Диаграмма состояний).**

Диаграмма состояний показывает переход объекта из одного состояния в другое.

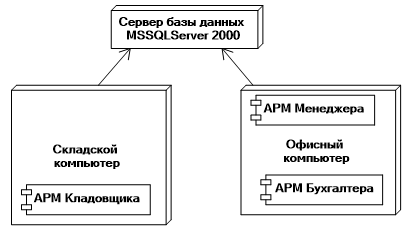
При построении используются прямоугольники с округленными

углами, представляющие состояние, через которое проходит объект в течение жизненного цикла. Стрелки указывают на переходы между состояниями объекта.



**Диаграмма кооперации**

Корпоративные приложения часто требуют для своей работы некоторой ИТ-инфраструктуры, хранят информацию в базах данных, расположенных где-то на серверах компании, вызывают веб-сервисы, используют общие ресурсы и т. д. В таких случаях полезно иметь графическое представление инфраструктуры, на которую будет развернуто приложение. Для этого и нужны диаграммы развёртывания, которые иногда называют диаграммами размещения.



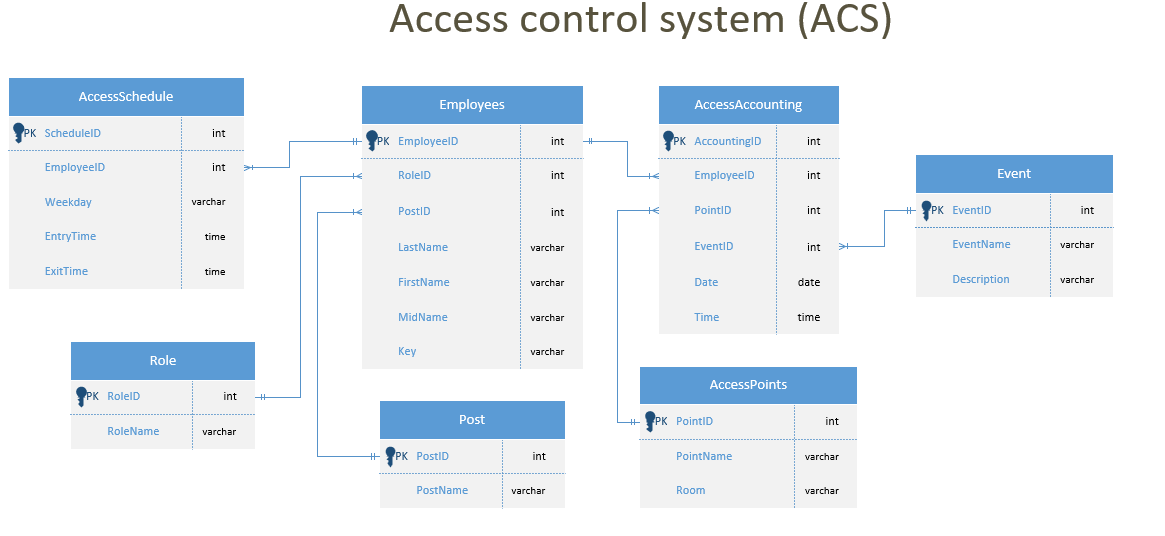
# **Логическая модели базы данных**

Логическая модель данных является начальным прототипом будущей базы данных. Логическая модель строится в терминах информационных единиц, но без привязки к конкретной СУБД. Более того, логическая модель данных необязательно должна быть выражена средствами именно реляционной модели данных. Основным средством разработки логической модели данных в настоящий момент являются различные варианты ER-диаграмм (Entity-Relationship, диаграммы сущность-связь) . Одну и ту же ER-модель можно преобразовать как в реляционную модель данных, так и в модель данных для иерархических и сетевых СУБД, или в постреляционную модель данных. Однако, т. к. мы рассматриваем именно реляционные СУБД, то можно считать, что логическая модель данных для нас формулируется в терминах реляционной модели данных.



# **Физическая модель данных**

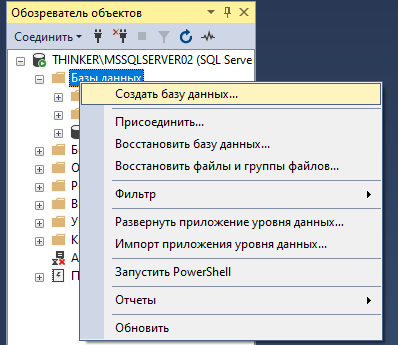
Физическая модель данных — это реализация логической модели данных, создаваемая администраторами и разработчиками баз данных. Она разрабатывается для определенных СУБД, технологий хранения и соединителей данных, чтобы по мере необходимости предоставлять данные через бизнес-системы пользователям. Это конечный результат всех остальных моделей



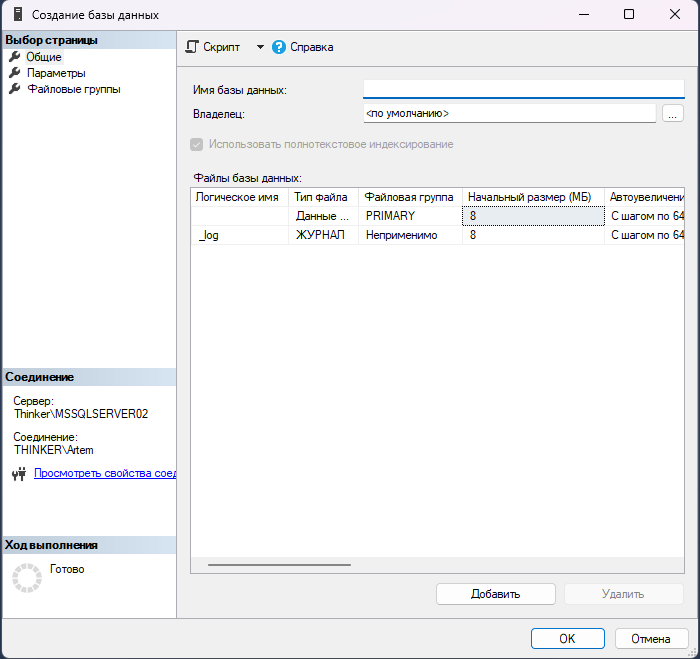
# **Создание Базы данных**

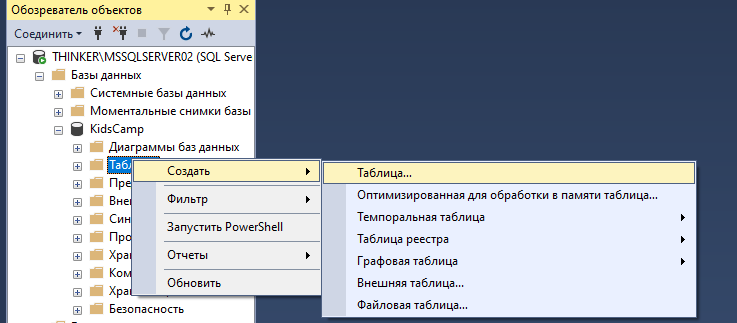
Мы открываем SQL Server Management Studio и нажимаем кнопку "Соединить".

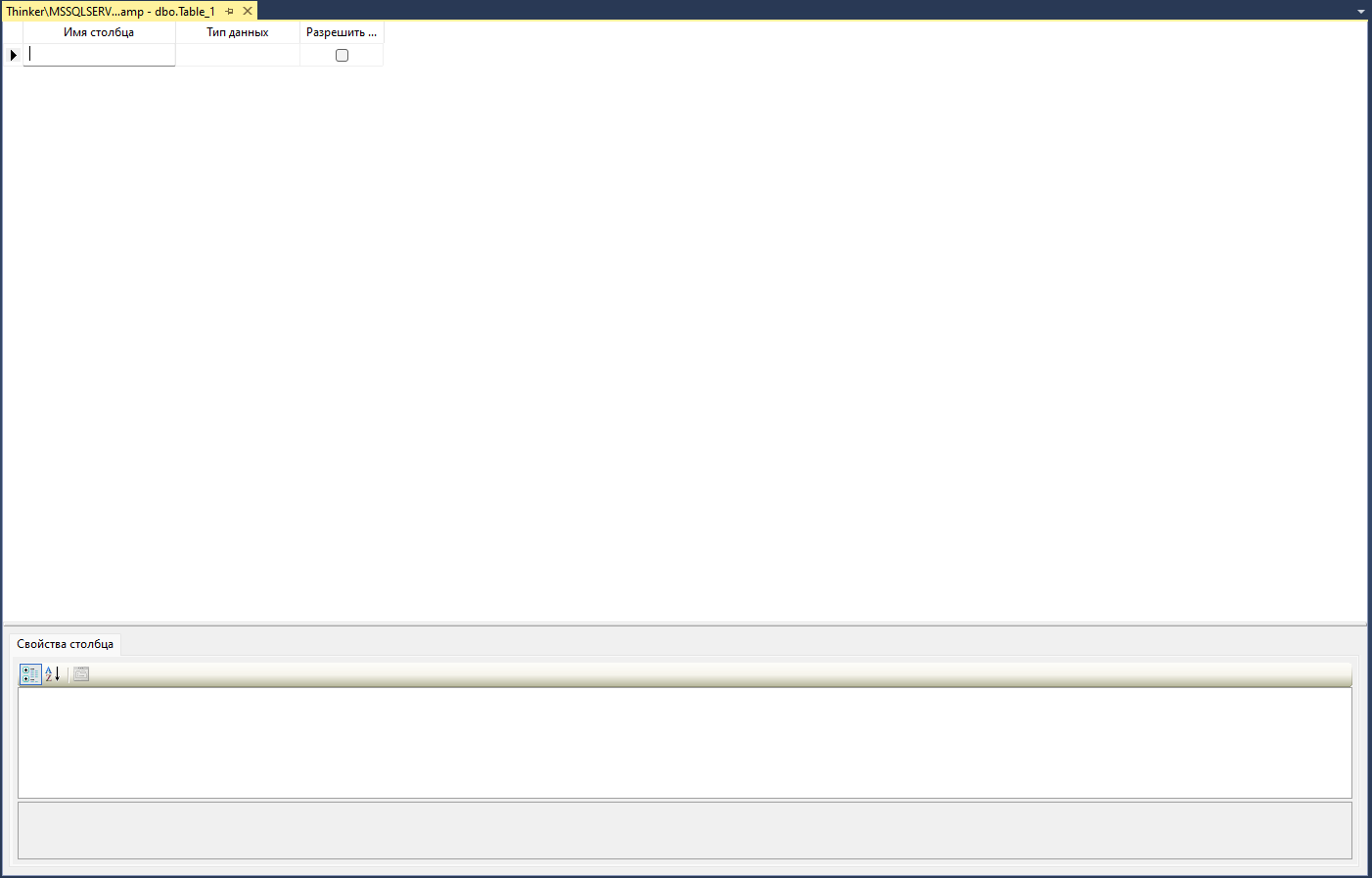
Создаем базу данных.



Присваиваем базе данных имя и нажимаем "OK". Она должна появиться в списке баз данных.



Раскрываем ее и создаем таблицу. 

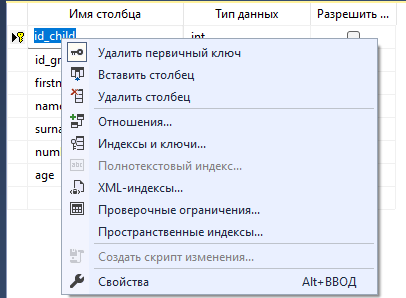
Открывается следующее окно.

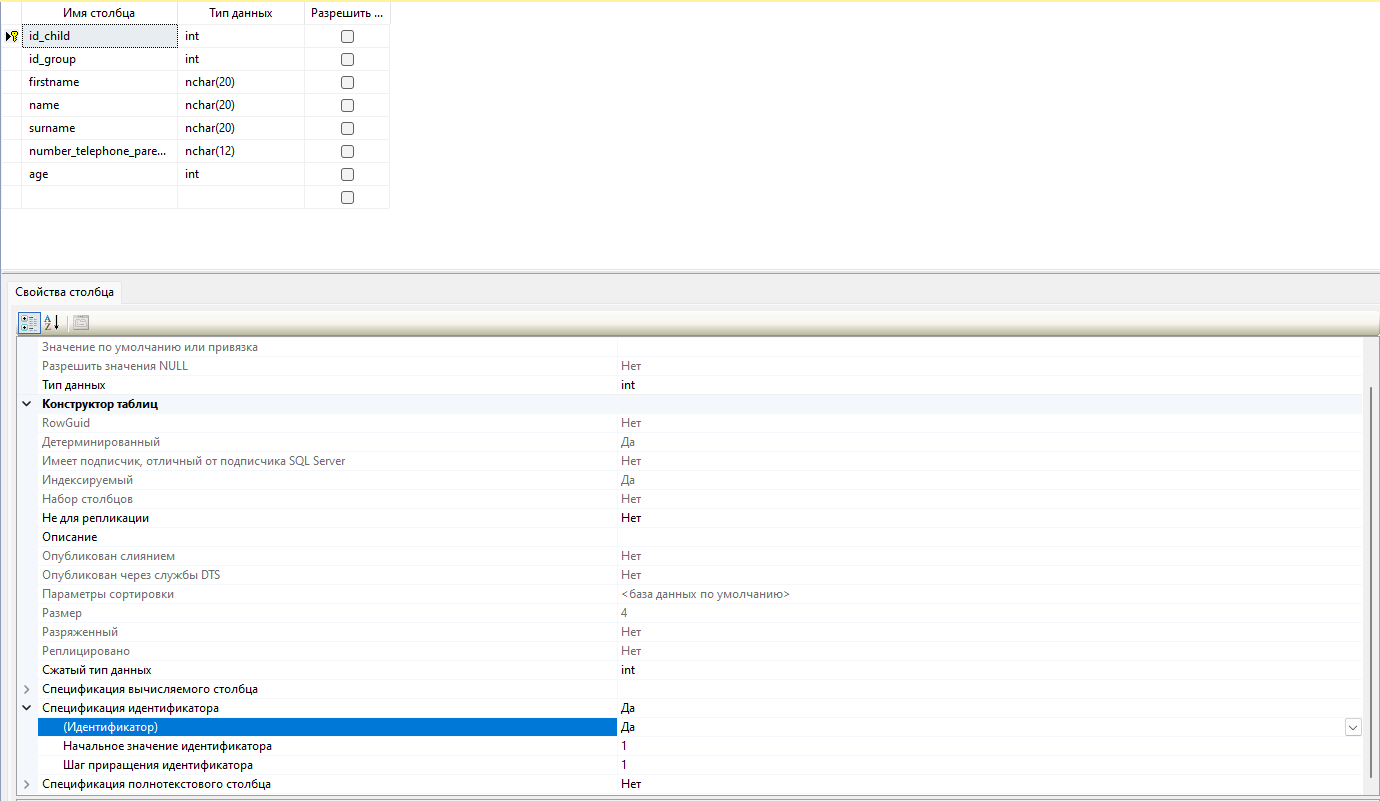
Мы задаем название для столбца и определяем значения, которые он будет хранить. Рекомендуется использовать английский язык для названий и разделять слова нижним подчеркиванием.

Основные типы данных:

* **Int**: хранит числа от -2147683648 до 2147683648.
* **Varchar()**: хранит строки длиной до указанного числа символов в скобках.
* **Datetime**: хранит значение даты и времени в формате ГГГГ-ММ-ДД ЧЧ:ММ:СС. Например, 2022-12-05 10:37:22. Диапазон значений от 1000-01-01 00:00:00 до 9999-12-31 23:59:59.
* **Bit**: хранит значение от 0 до 16. Может выступать аналогом булевого типа в языках программирования (в этом случае значению true соответствует 1, а значению false - 0).
* С дополнительными типами можно ознакомиться на сайте: <https://metanit.com/sql/sqlserver/3.3.php>

В таблице всегда должен присутствовать столбец "ID". Нам нужно задать его как первичный ключ.

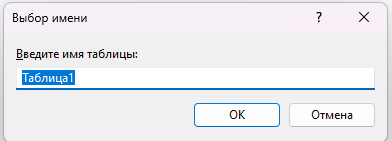


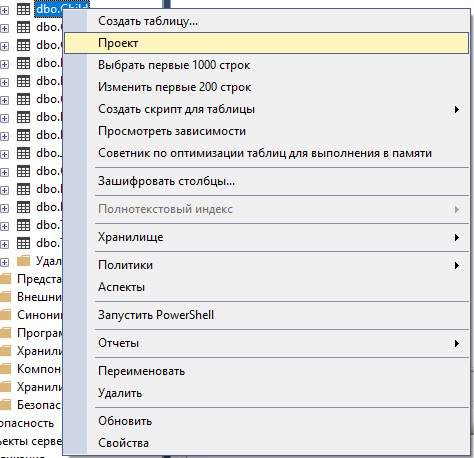
Мы также можем задать идентификатор, чтобы его значение присваивалось автоматически.

И нужно выключить разрешения на пустые значения.

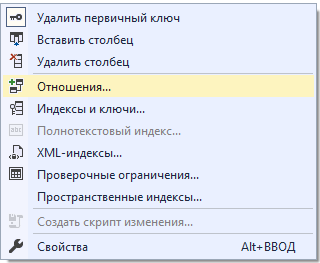


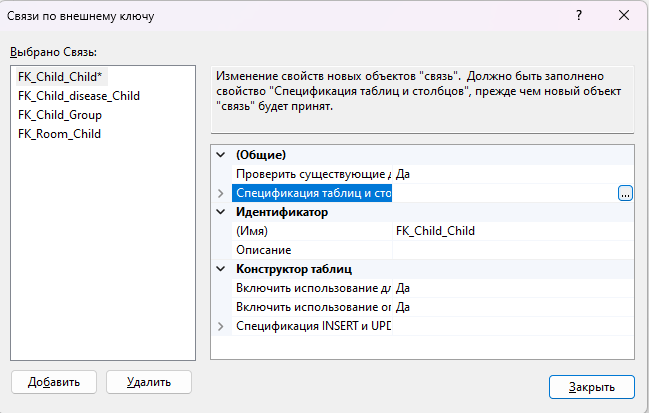
После создания таблицы мы сохраняем ее и присваиваем ей имя с помощью комбинации клавиш Ctrl + S.



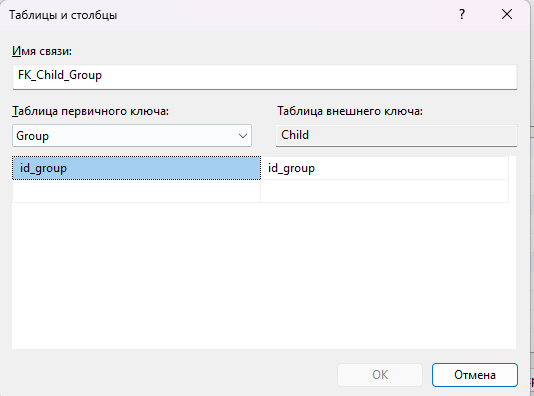
Для связывания таблиц нам нужно открыть проект таблицы.

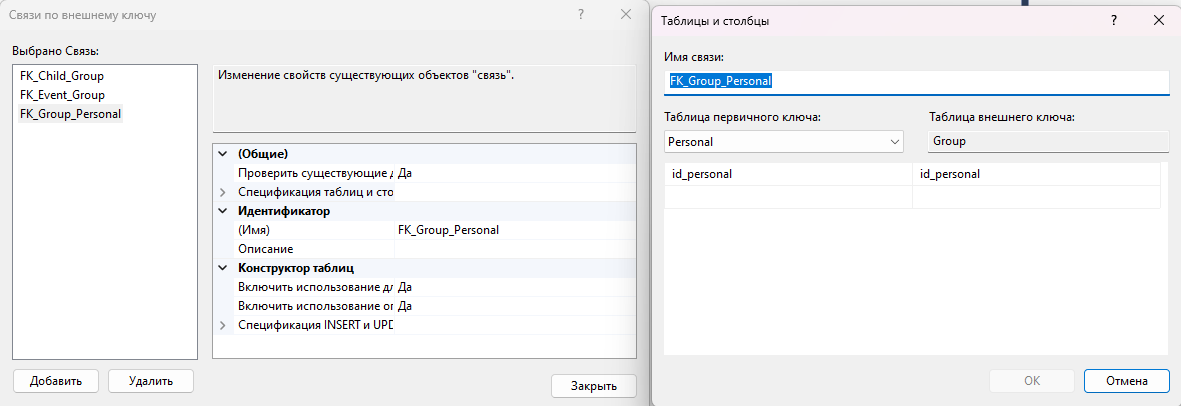
Мы кликаем правой кнопкой мыши внутри проекта таблицы и выбираем "Отношения".



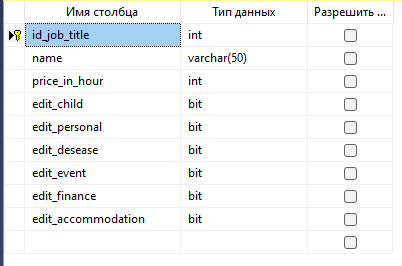
Нажимаем "Добавить", открываем спецификацию таблиц и столбцов.

В открывшемся окне мы выбираем таблицу первичного ключа (таблицу, с которой мы связываем) и столбец, который будем связывать (ID таблицы), а также столбец, с которым будем связывать, и нажимаем "OK". После этого сохраняем.

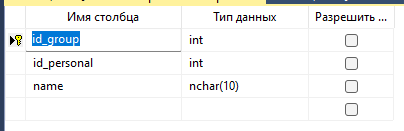


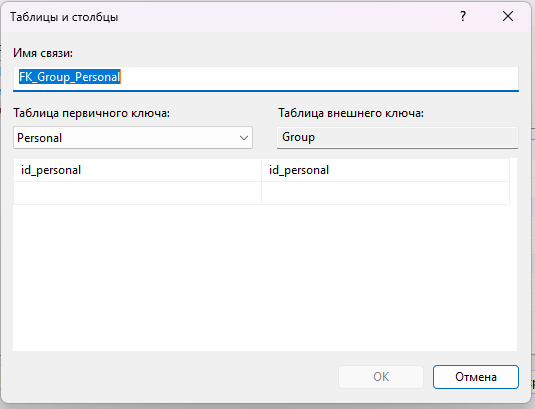
В таблице, которую мы связали, также должна появиться связь.

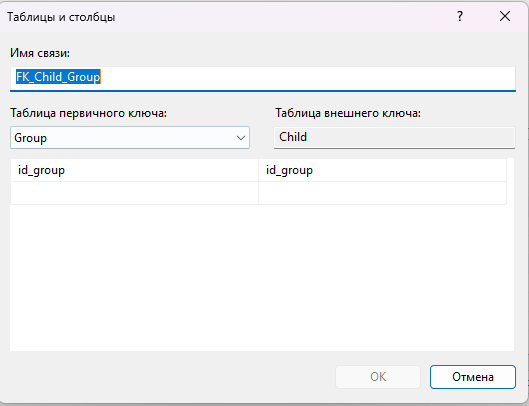
И надо создать окно с должностями, в котором будут хранится разрешения на редактирования данных, оно пригодится в будущем для разделения прав

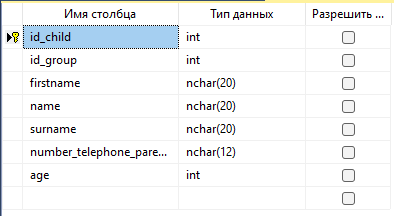


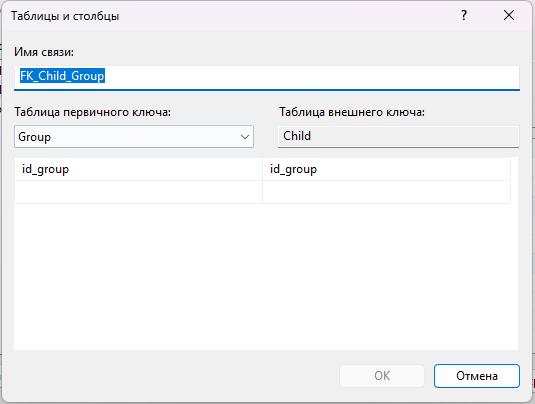
**Примеры таблиц и их связей:**











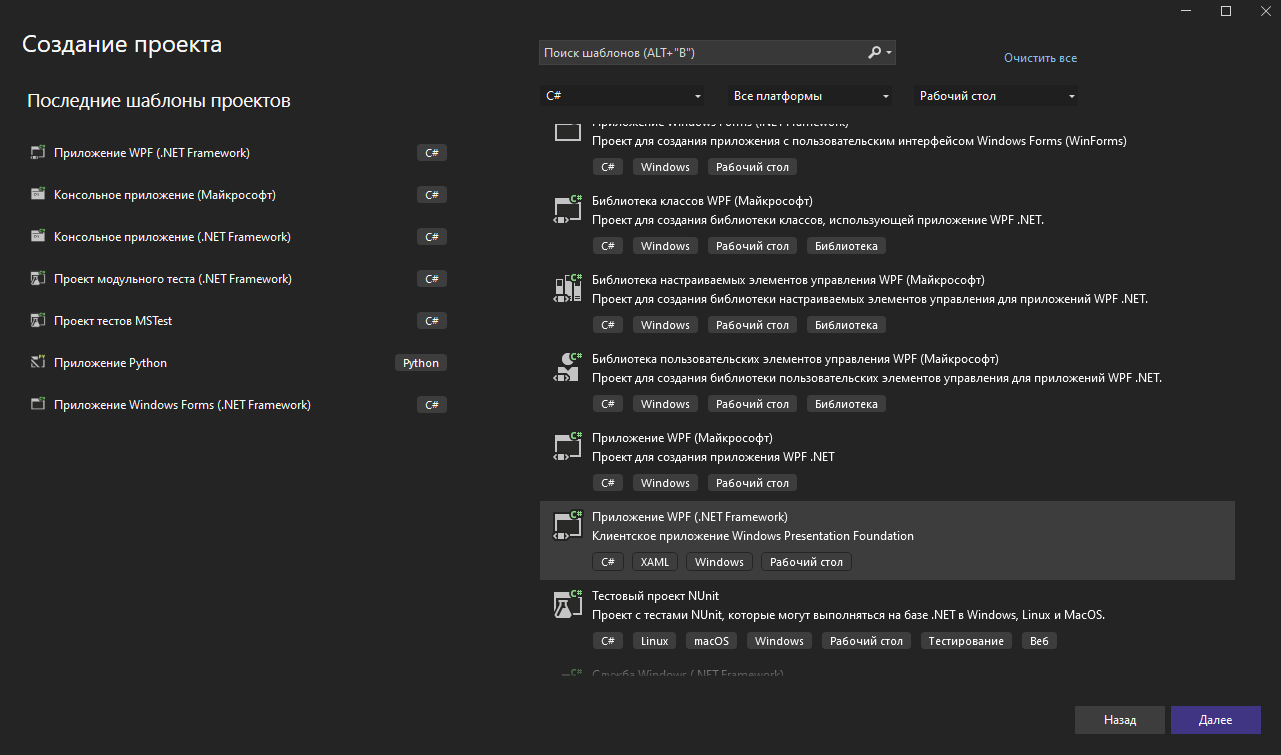
# **Реализация информационной системы**

При написании приложения будем использовать паттерн MVVM

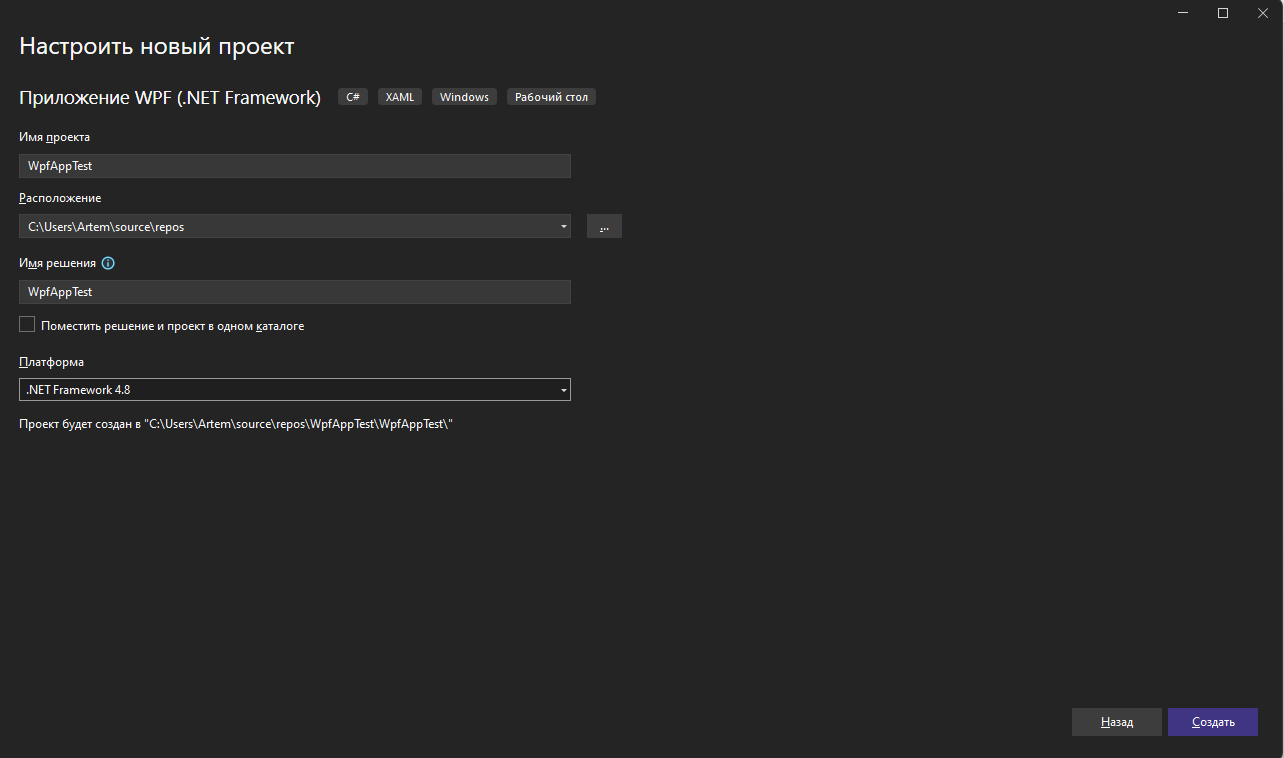
Подробнее про него:

* <https://metanit.com/sharp/wpf/22.1.php>
* <https://habr.com/ru/articles/338518/>
* https://vk.com/@73240119-c-postgresql-osnovy-entity-framework-core)

Открываем Visual Studio и создаем проект



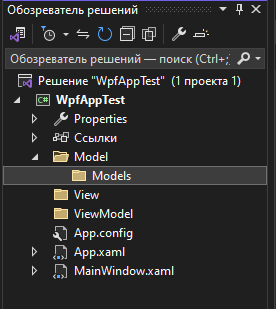
Даем ему имя и выбираем последнею версию .Net Framework и жмем создать



осле создания проекта создадим следующую структуру папок:

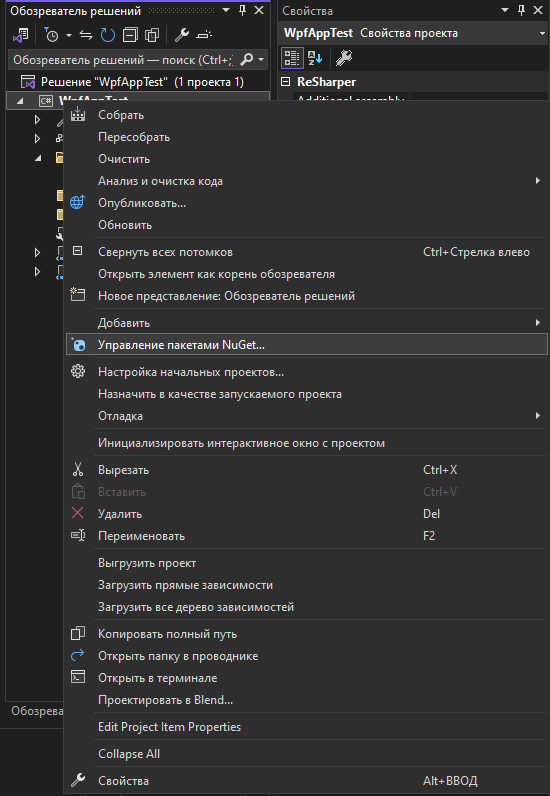
* Model
* View
* ViewModel

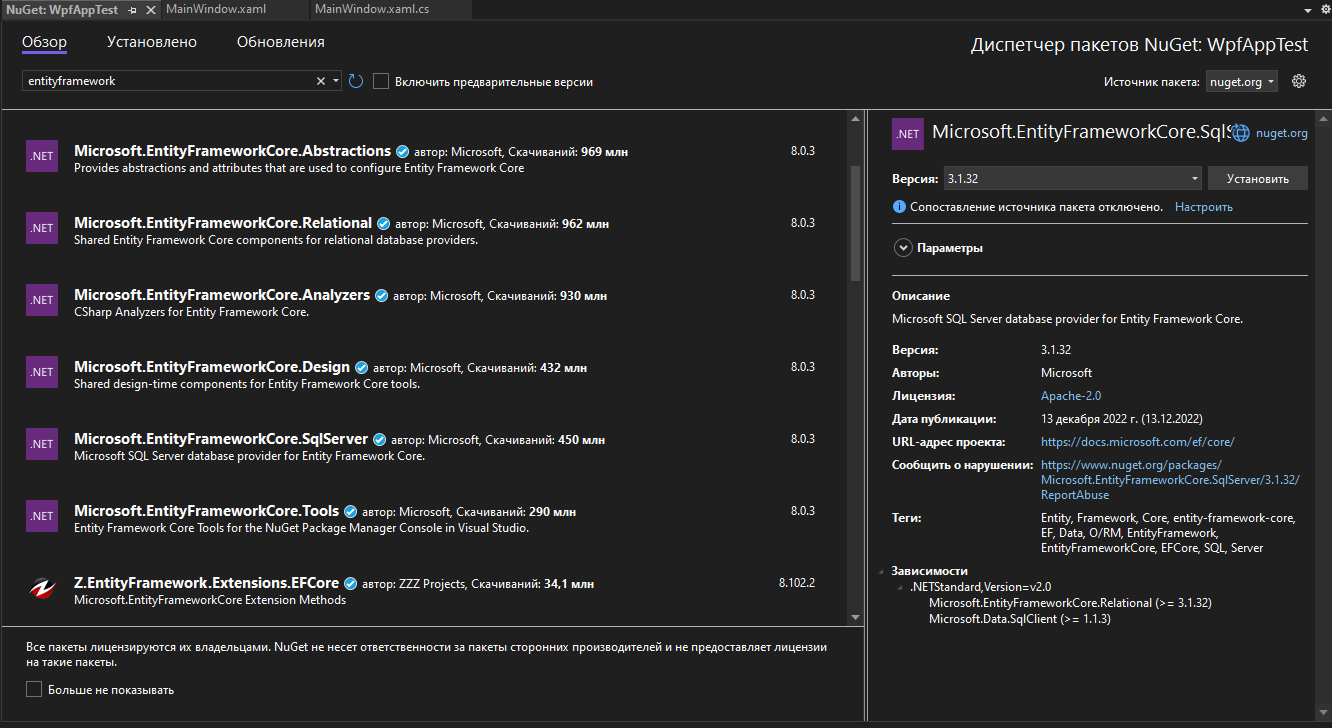
Внутри папки Model создадим еще одну папку Models.

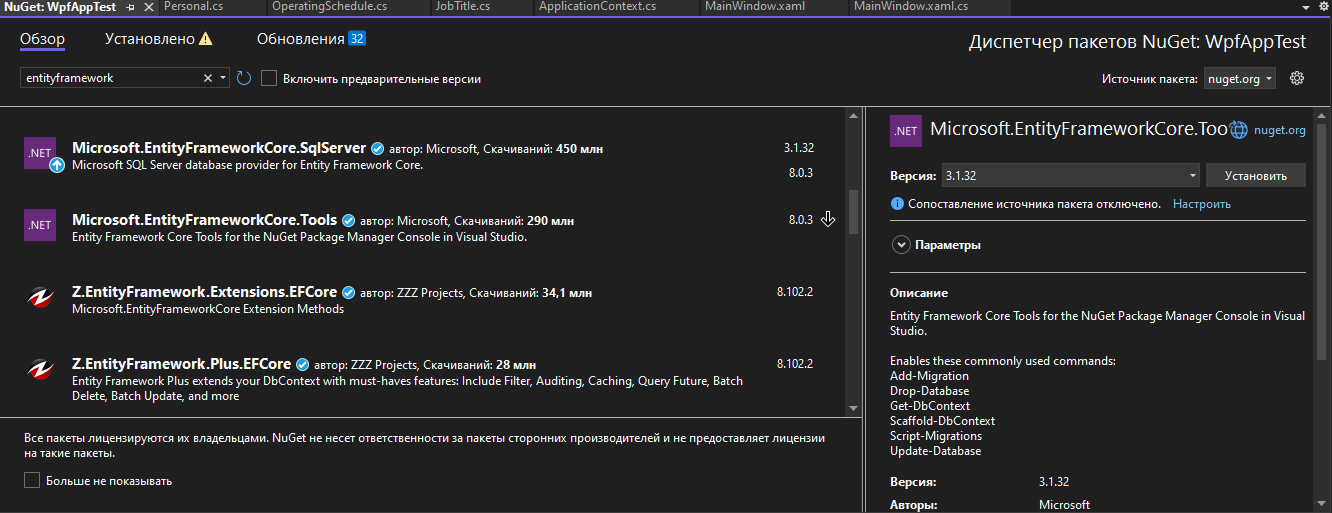


**Model**

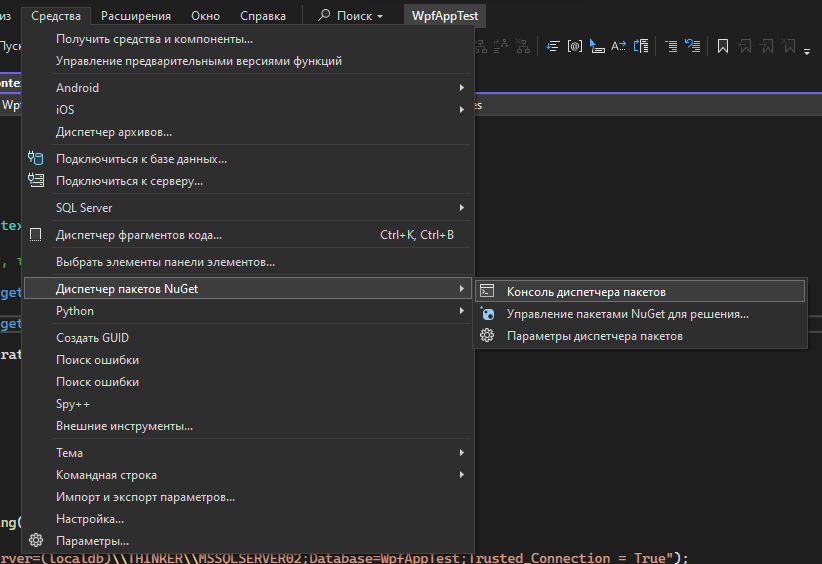
Откроем окно управления пакетами NuGet.



На вкладке "Обзор" найдем и установим пакеты **Microsoft.EntityFrameworkCore.SqlServer** версии 3.1.32.

и **Microsoft.EntityFrameworkCore.Tools** версии 3.1.32.

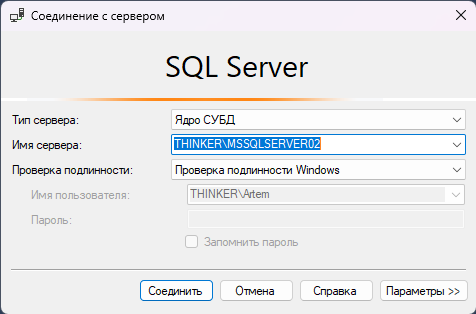
Откроем диспетчер пакетов



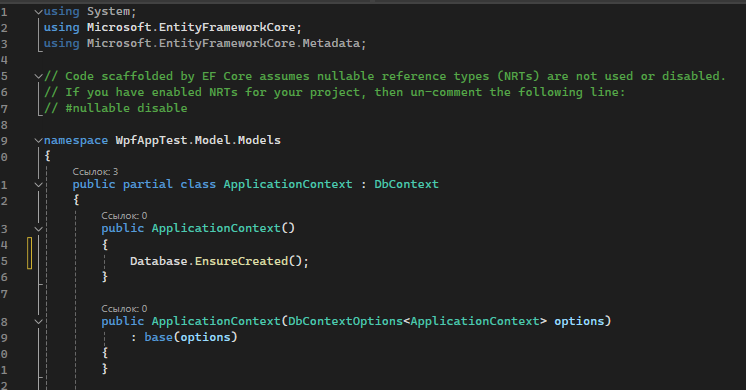
и введем команду:



Scaffold-DbContext "Data Source=THINKER\MSSQLSERVER02;Initial Catalog=KidsСamp;Integrated Security=True; " Microsoft.EntityFrameworkCore.SqlServer -OutputDir Model/Models -Context ApplicationContext

Где **THINKER\MSSQLSERVER02** - имя сервера, а **KidsСamp** - название базы данных. **Model/Models** - это папка, в которой будут созданы классы.

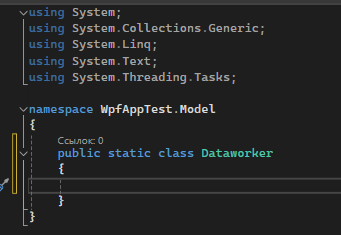
После выполнения команды должны появиться классы. В классе **ApplicationContext** добавим строку **Database.Encured**.



Теперь создадим статический класс **DataWorker** в папке Model для выполнения операций CRUD:

Подробнее про Crud операции:

* <https://www.youtube.com/watch?v=i38HKuuECCM&t=16s&ab_channel=RuslanShishmarev>

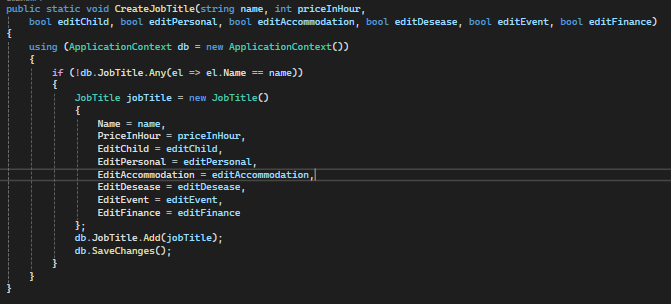


Теперь создадим методы добавления, удаления, редактирования для класса JobTitle



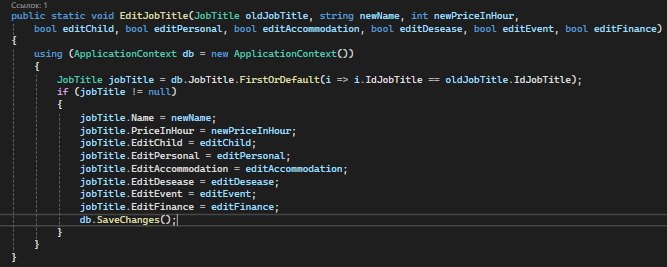
Создадим метод добавления **CreateJobTitle** для класса **JobTitle**. Параметры метода: **string name, int priceInHour,** **bool editChild, bool editPersonal, bool editAccommodation, bool editDesease, bool editEvent, bool editFinance**

1. Используем конструкцию **using** для безопасного соединения с базой данных.
2. Проверяем, существует ли в базе данных должность с таким именем.
3. Если нет, создаем новую должность, добавляем ее в таблицу и сохраняем изменения.



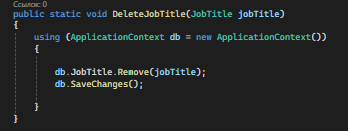
Создадим метод редактирования **EditJobTitle**. Параметры метода: **JobTitle oldJobTitle, string newName, int newPriceInHour, bool editChild, bool editPersonal, bool editAccommodation, bool editDesease, bool editEvent, bool editFinance**

1. Используем конструкцию **using** для безопасного соединения с базой данных.
2. Находим в базе данных должность, которую будем редактировать.
3. Если должность найдена, изменяем ее имя и зарплату в час, а затем сохраняем изменения.



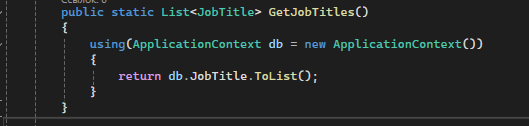
Создадим метод удаления **DeleteJobTitle**. Параметр метода: **JobTitle**.

1. Используем конструкцию **using** для безопасного соединения с базой данных.
2. Удаляем должность и сохраняем изменения.

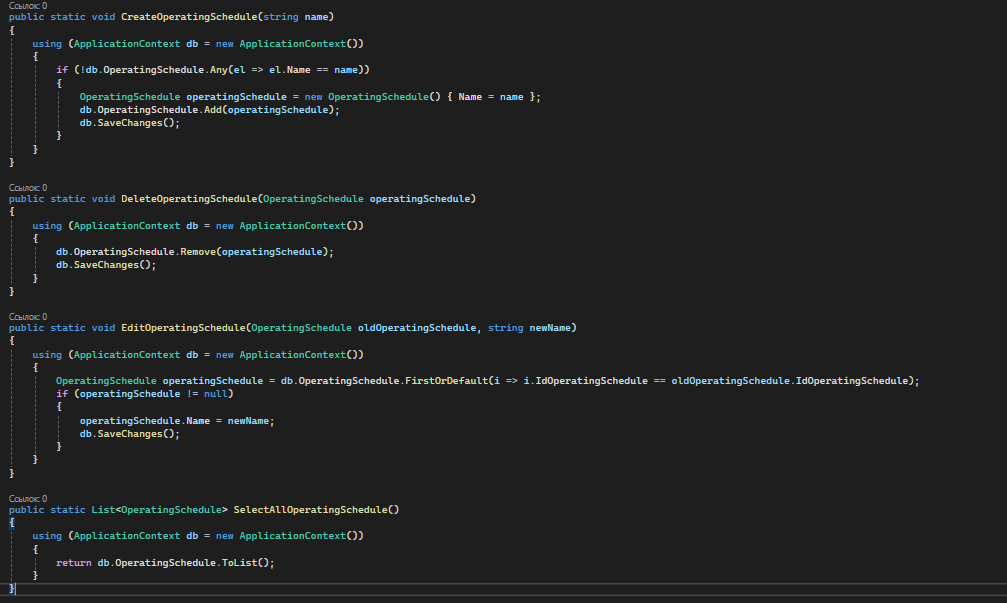


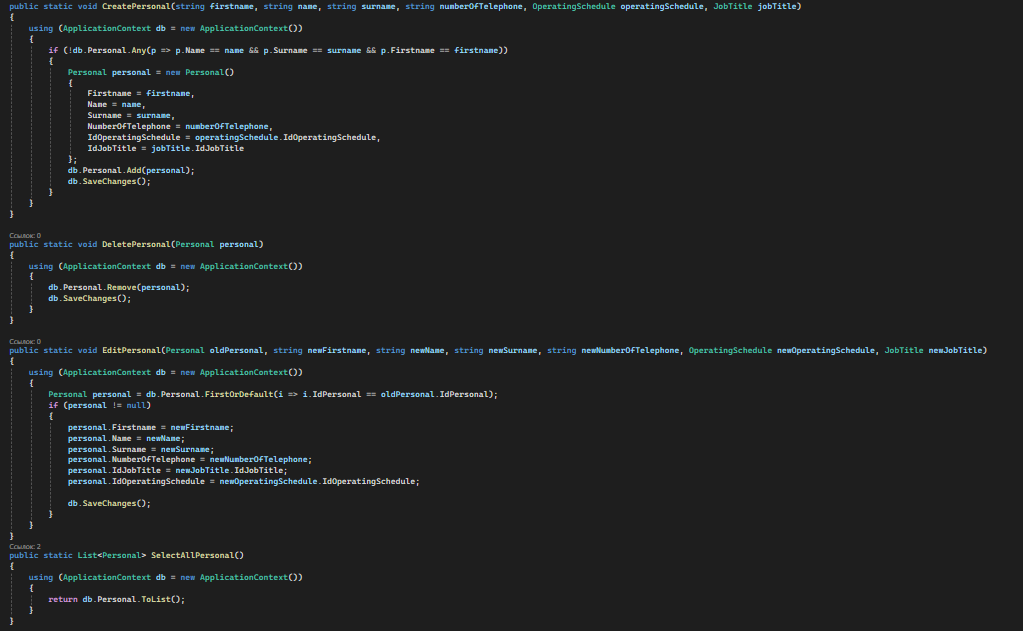
Создадим метод **GetJobTitles** для получения всех элементов таблицы должностей.

1. Используем конструкцию **using** для безопасного соединения с базой данных.
2. Возвращаем все элементы таблицы должности.
3. Возвращаем все элементы таблицы должности



Еще примеры таких методов





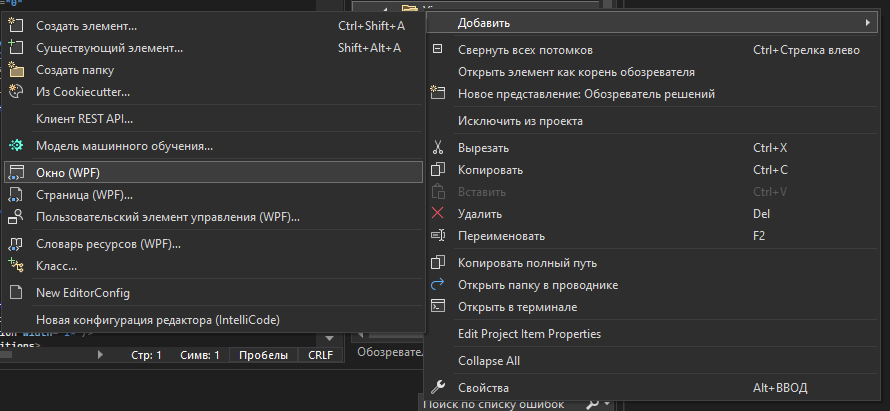
Такие методы необходимо создать для каждой таблицы

**View**

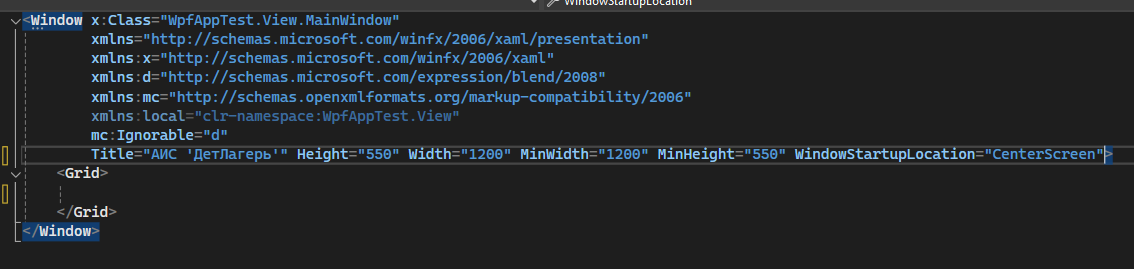
В этой части, будет создан весь интерфейс.

Начнем с главного окна.

Создаем окно в папке view и даем ему название



У окна задаем следующие свойства



Title – имя окна

Height – высота окна

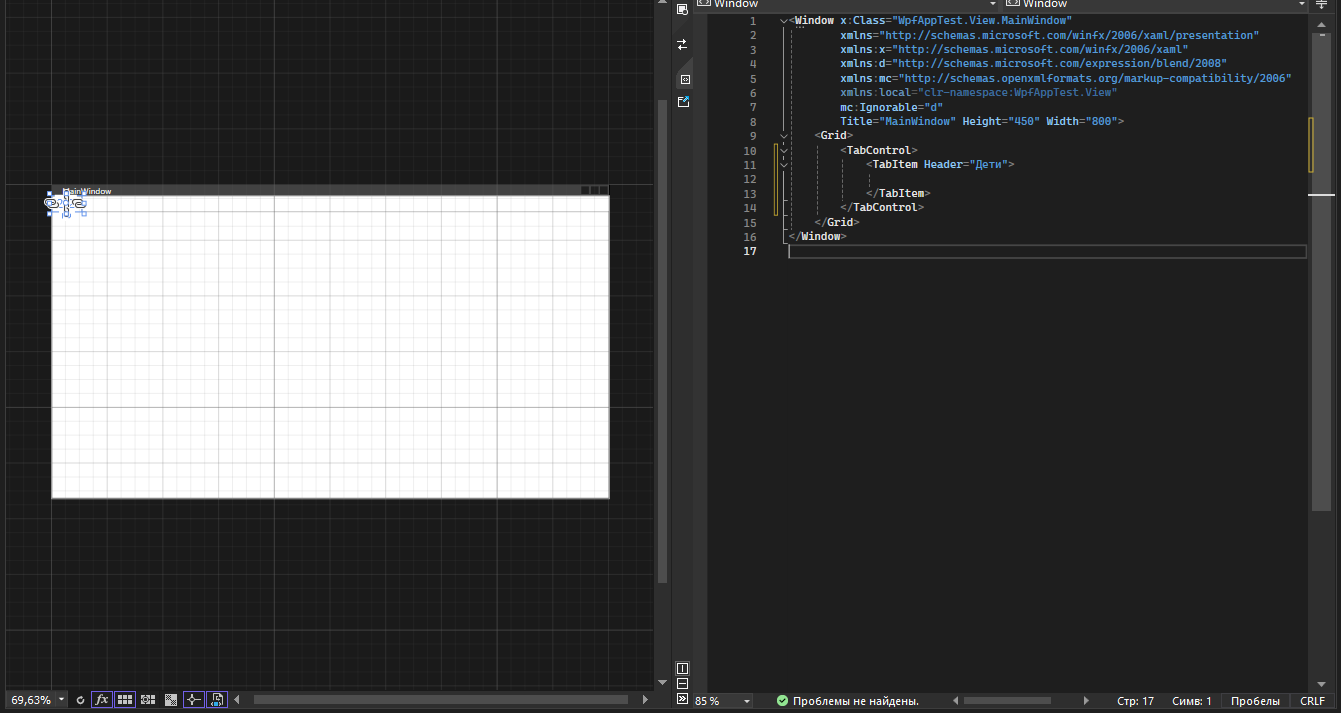
Width – ширина окна

MinWidht – минимальная ширина окна

MinHeight – минимальная высота окна

WindowStartupLocation – позиция появления окна

Внутри окна создаем элемент



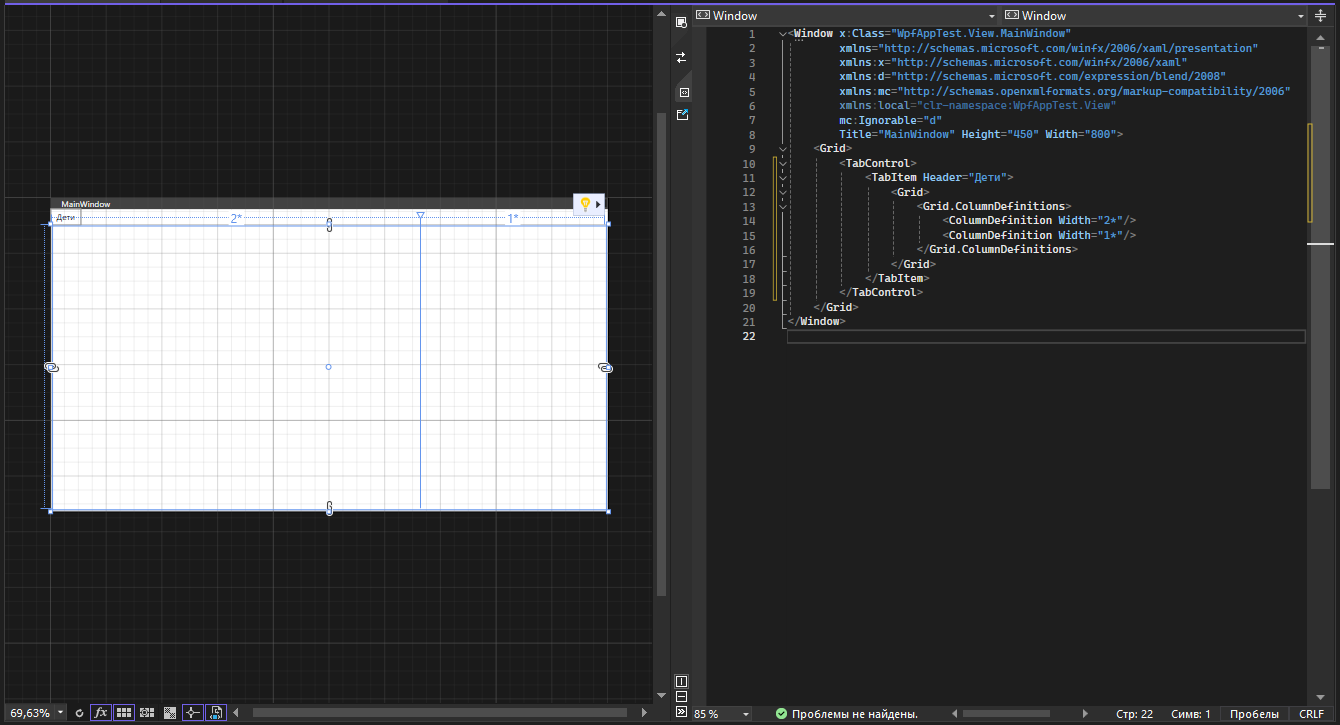
TabControl – это контейнер в котором содержаться вкладки

TabItem – вкладка как в браузере

Header – название вкладки

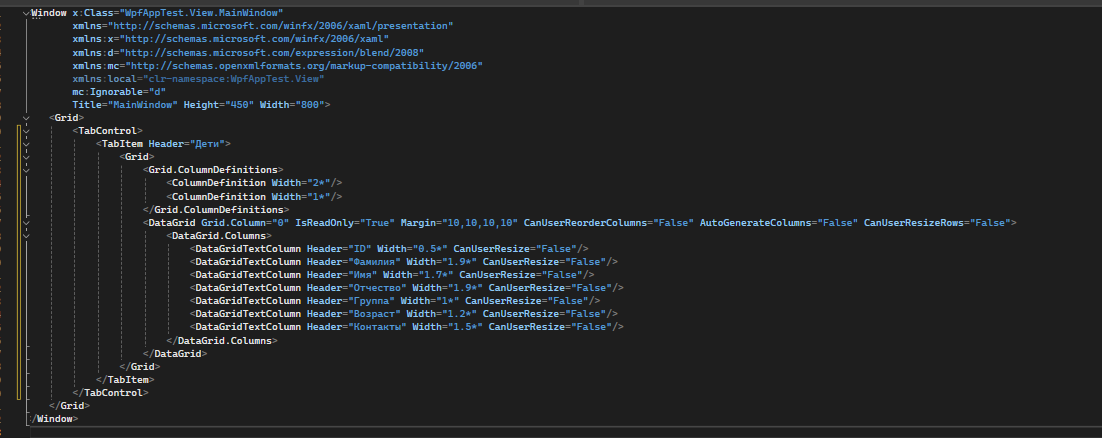
Внутри TabItem создаем Grid это сетка разметки, она нужна для правильного расположения элементов.

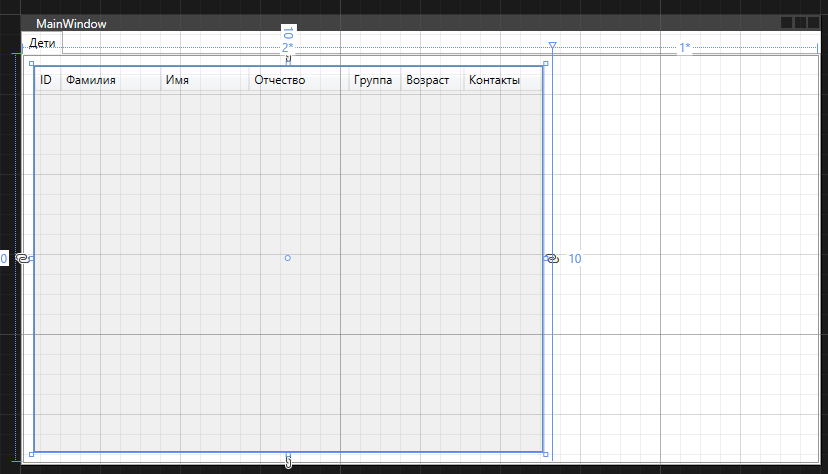
И создаем два столбца



With - это ширина столбца. 1\* и 2\* — это относительные размеры, это значит размер первый столбец будет занимать 2/3 ширины окна, а второй 1/3.

Далее создаем таблицу в первом столбце





**DataGrid** – это таблица:

* **Grid.Column**="0" – свойство, устанавливающее таблицу в первом столбце Grid
* **IsReadOnly**="True" – свойство определяющее, может ли пользователь изменять данные в таблице.
* **Margin**="10,10,10,10" – отступы по краям: слева, сверху, справа, снизу соответственно
* **CanUserReorderColumns**="False" – свойство определяющее, может ли пользователь изменять порядок отображения столбцов, перетаскивая их заголовки мышью.
* **AutoGenerateColumns**="False" – свойство определяющее, могут ли автоматически создаваться столбцы, оно пригодится в будущем
* **CanUserResizeRows**="False" – свойство определяющее, может ли пользователь изменять размеры строк

**DataGridTextColumn** – столбец таблицы, в котором будет текст:

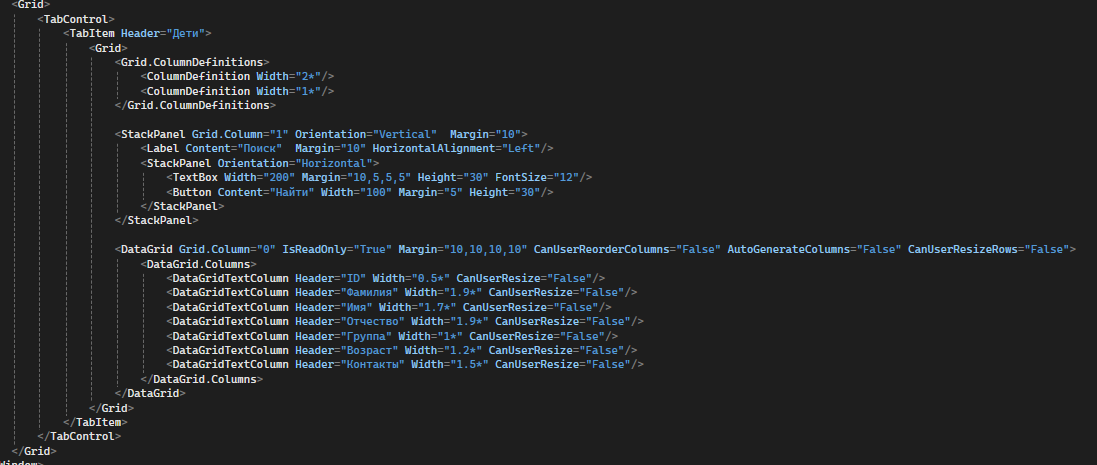
* **CanUserResize**="False"- свойство задающее, может ли пользователь менять размеры

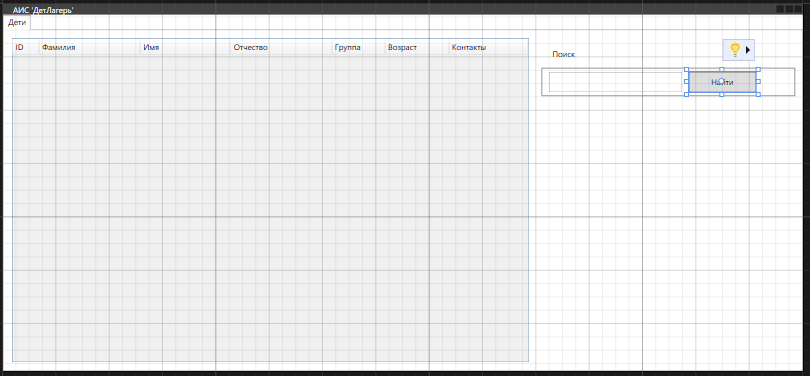
Далее создаем **StackPanel** – это контейнер, в котором хранятся другие элементы



**Orientation**="Vertical" – свойство задающее, в какой ориентации будут добавляться элементы помещённые в StackPanel

Внутри StackPanel создаем





**Label** – это надпись на окне:

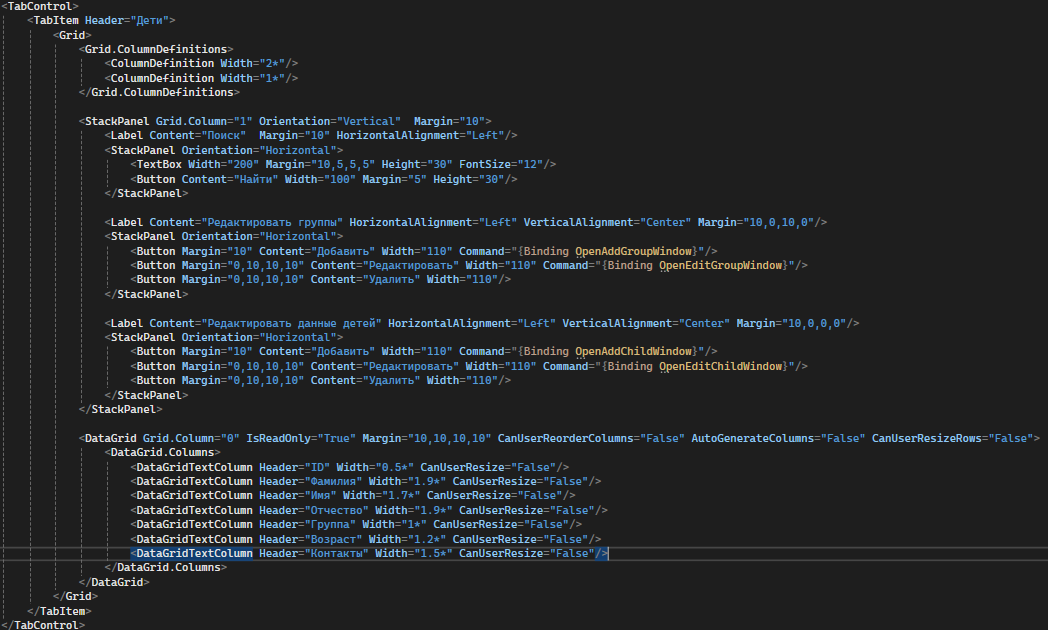
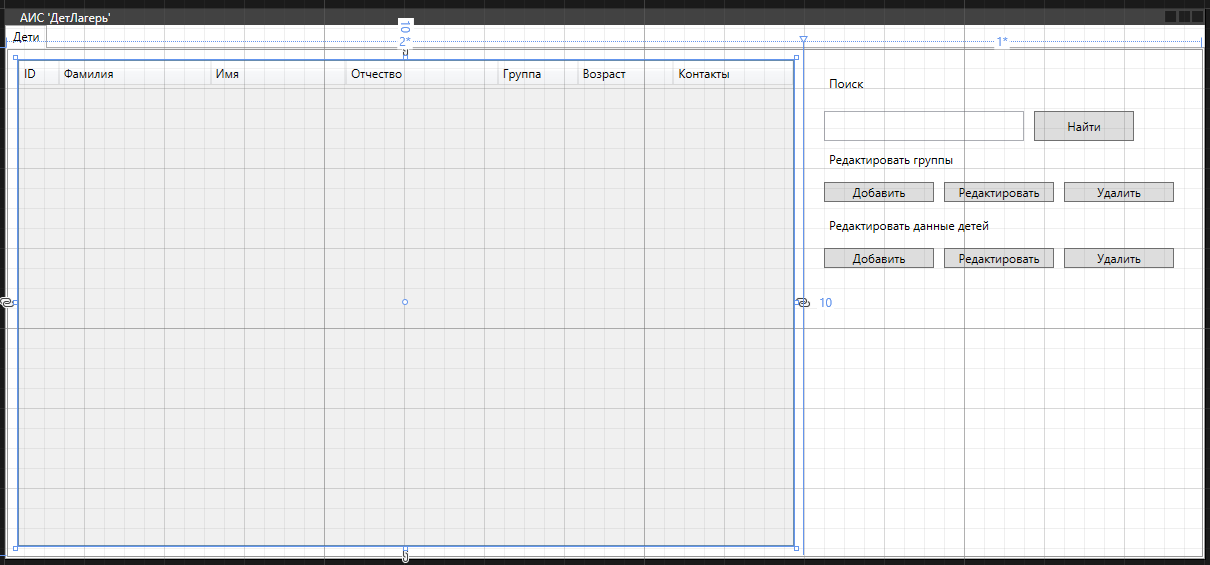
* **Content** – это текст, который будет отображаться в элементе
* **HorizontalAlignment** – горизонтальное вырывание элемента

**TextBox** – элемент ввода текста:

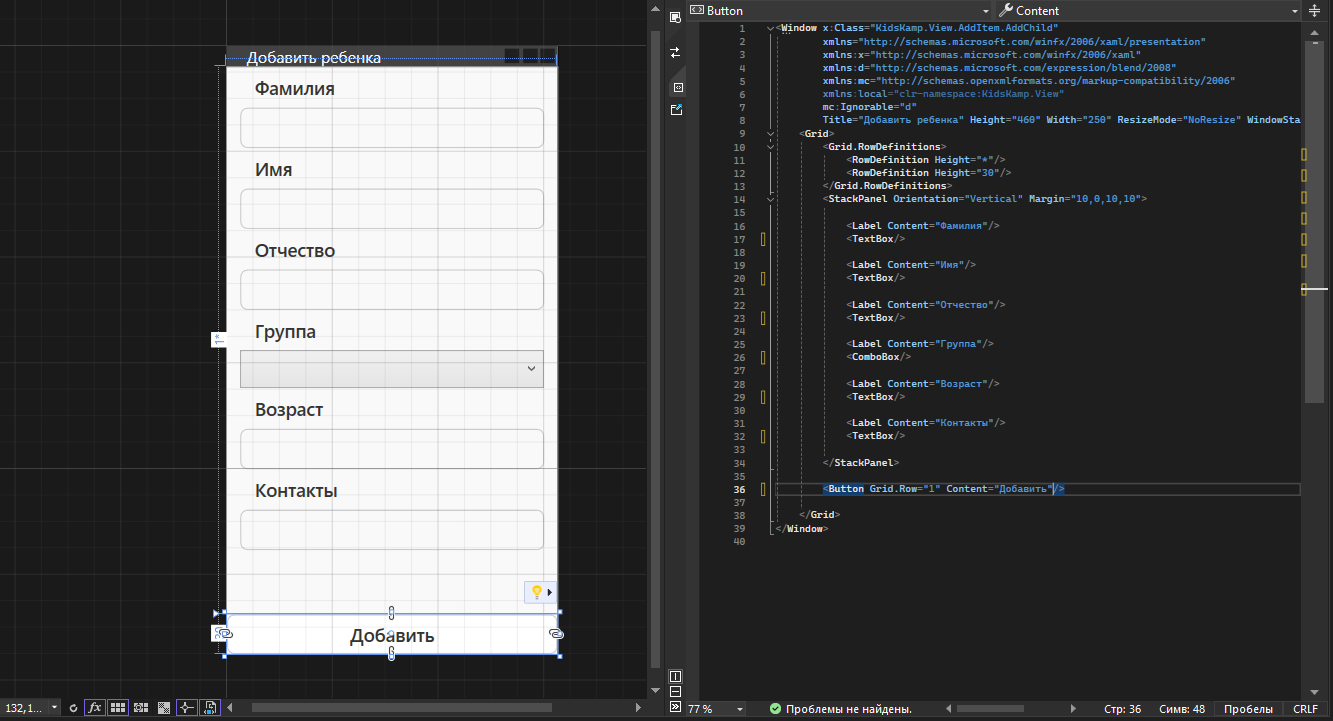
* **Width** – ширина элемента
* **Height** – высота элемента
* **FontSize** – размер текста

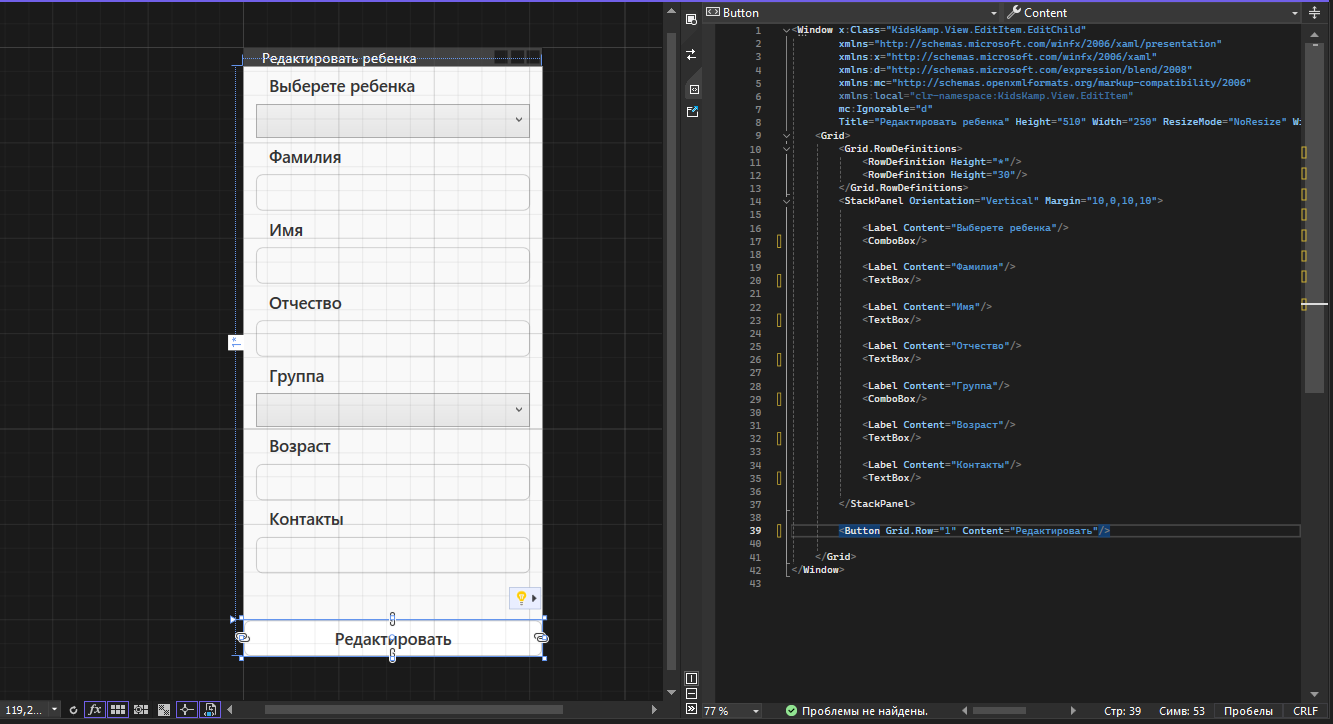
**Button** – кнопка

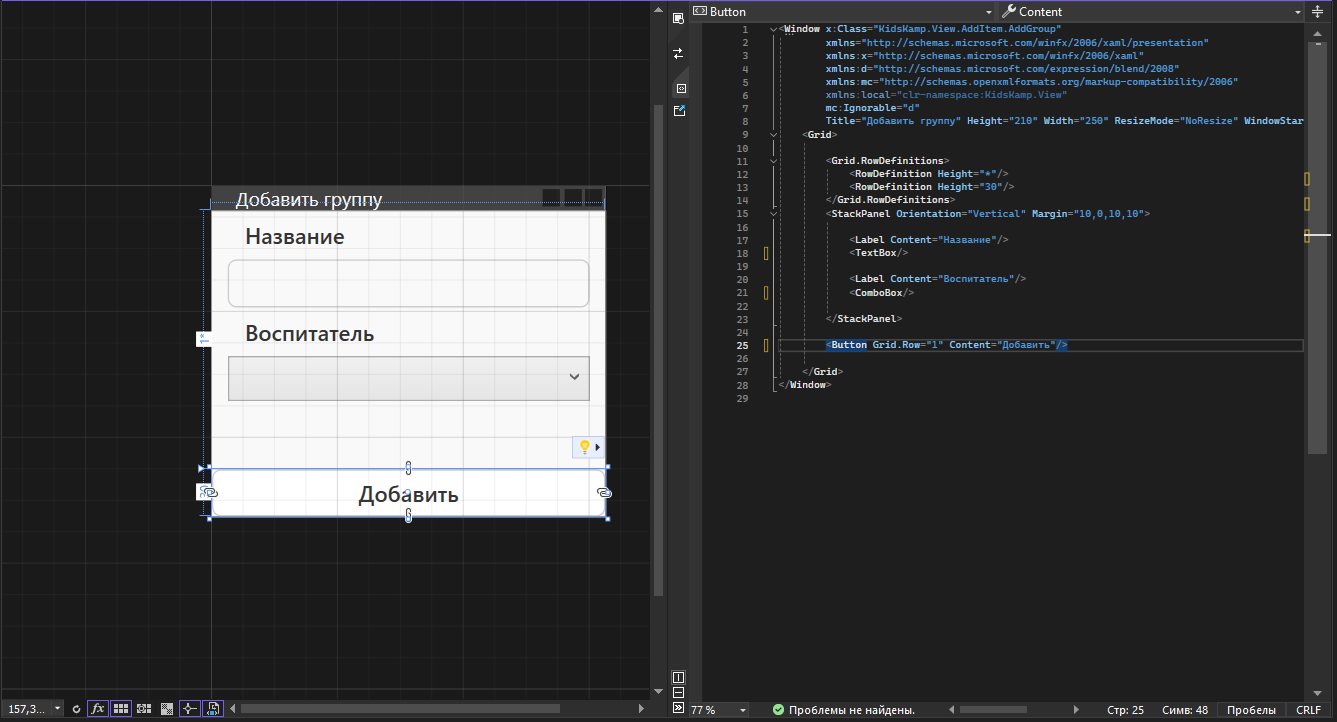
Далее создаем следующие элементы

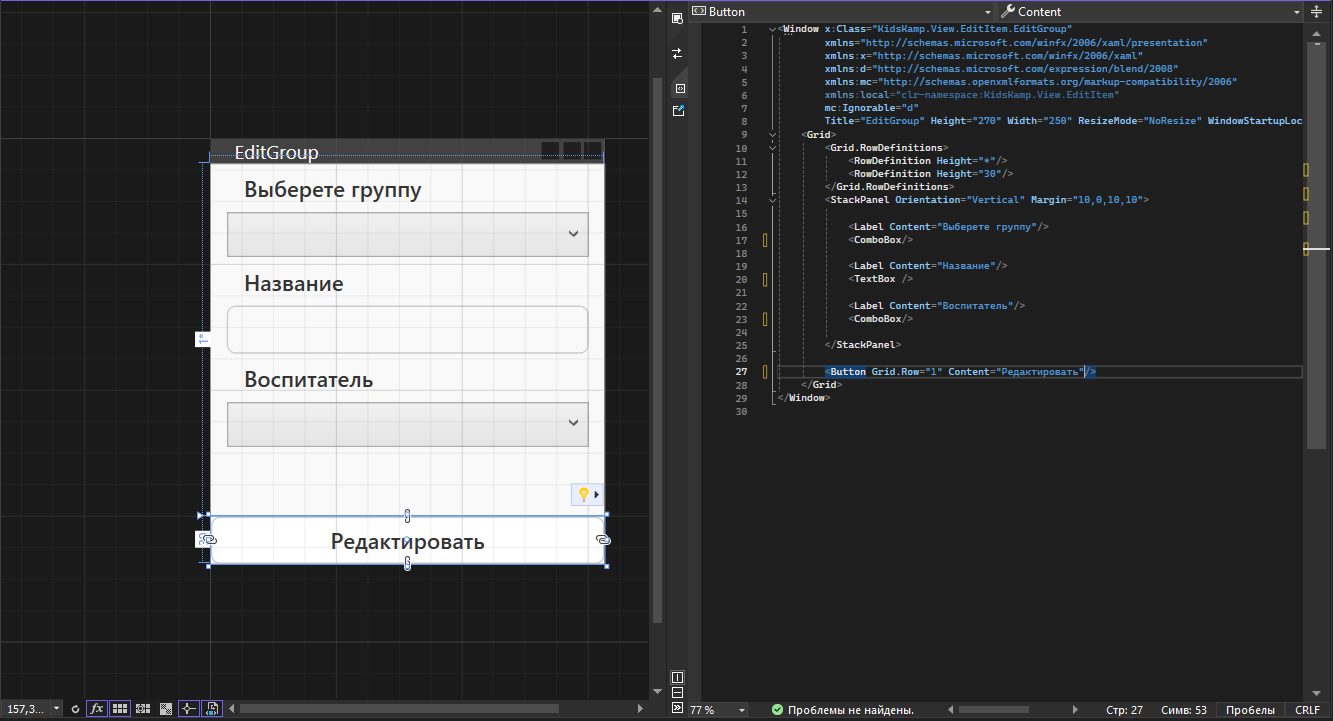
 

Далее создаем окна добавления, редактирования группы, окна добавления, редактирования детей и окно удаления

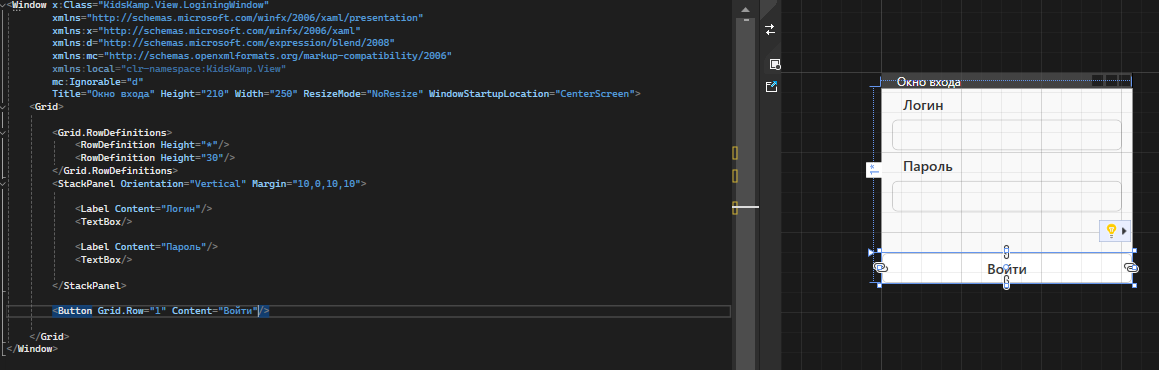






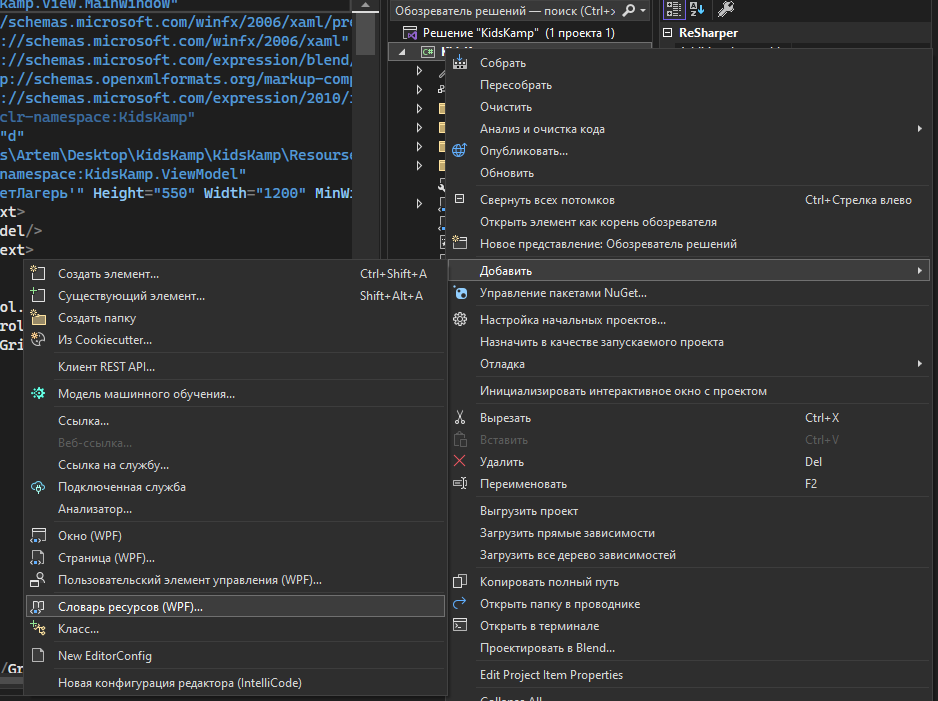


И окно входа



**Создание стилей для элементов интерфейса**:

Создайте новый файл XAML в папке **View**, например, **Styles.xaml**, и определите стили для различных элементов



Пример, для меток (**Label**):

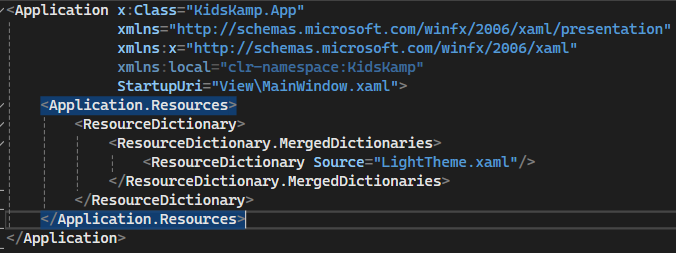
**Style** – это контейнер для свойств:

* **TyrgetType** – тип элемента

**Setter** - свойство:

* Property – название свойства
* Value – значение по умолчанию, которое будет применяться к свойству

Откройте файл **App.xaml** и добавьте следующую секцию **ResourceDictionary** для подключения файла стилей:

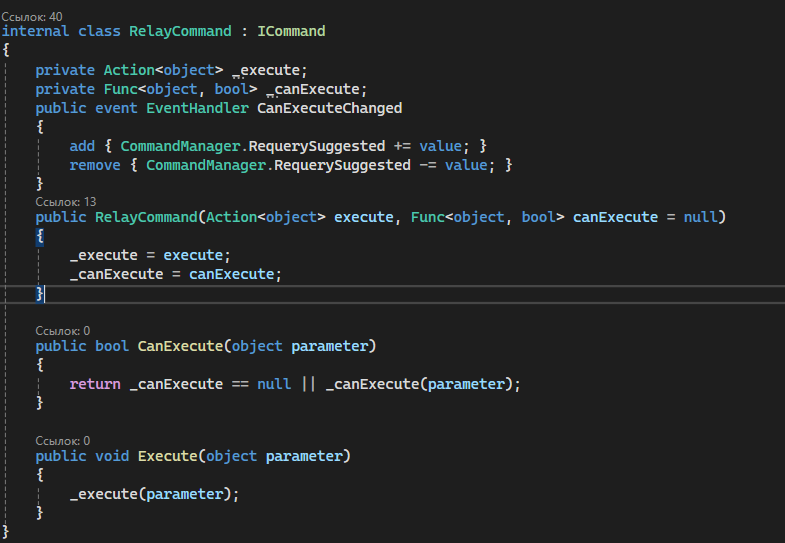


**Source** – это путь к файлу

И еще тут можно изменить стартовое окно

**StartupUri** – путь к стартовому окну

**ViewModel**

Приступим к созданию ViewModel для соединения View и Model в вашем WPF приложении. Создаём класс RelayCommand:

internal class RelayCommand : ICommand

{

private Action<object> \_execute;

private Func<object, bool> \_canExecute;

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)

{

\_execute = execute;

\_canExecute = canExecute;

}

public bool CanExecute(object parameter)

{

return \_canExecute == null || \_canExecute(parameter);

}

public void Execute(object parameter)

{

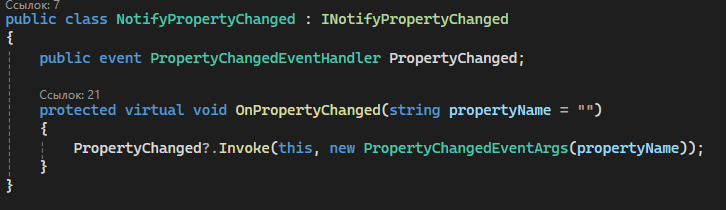
\_execute(parameter);

}

}

Этот класс нам пригодится в будущем

Теперь создаем класс NotifyPropertyChanged и пишем там следующий код:



public class NotifyPropertyChanged : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged(string propertyName = "")

{

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

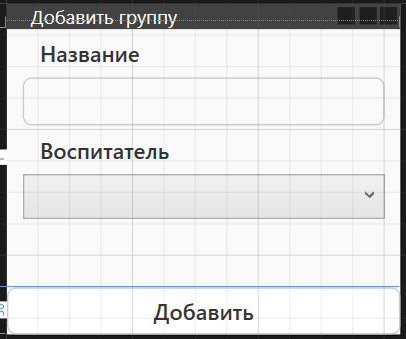
}

}

Теперь создадим класс AddGroupViewModel и наследуемся от класса NotifyPropertyChanged

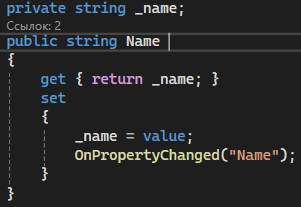


У нас есть следующее окно:



Чтобы добавить группу, нам надо знать ее название и воспитателя

Значит в классе AddGroupViewModel создаем свойство Name



Подробнее про свойства: https://metanit.com/sharp/tutorial/3.4.php

Далее нам нужно создать список всех сотрудников для Combobox элемента

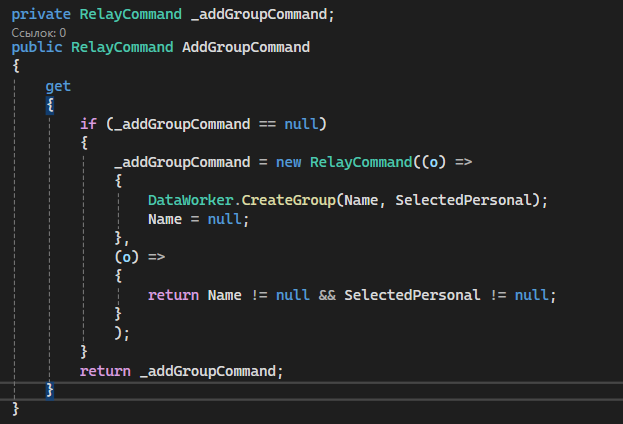


DataWorker.SelectAllPersonal(); - возвращает всех сотрудников из бд

Далее нам нужно создать свойство выбранного сотрудника



Теперь надо создать команду, которая будет выполняться при нажатии на кнопку сохранить



(o) =>

{

DataWorker.CreateGroup(Name, SelectedPersonal);

Name = null;

},

Этот блок кода выполняется при выполнении команды

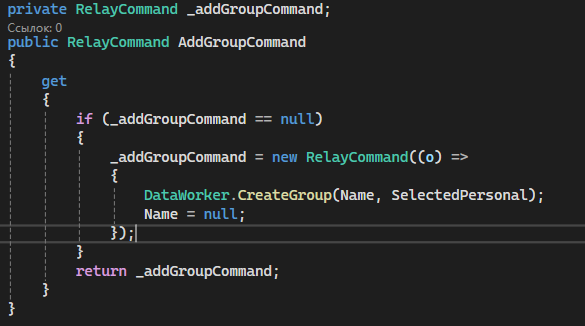
(o) =>

{

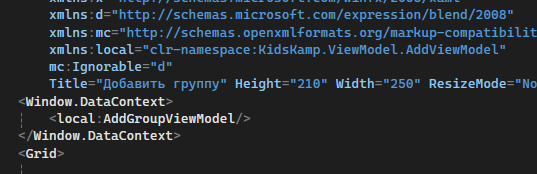
return Name != null && SelectedPersonal != null;

}

Этот блок смотрит, когда команду можно выполнить, в примере если имя будет пустое или выбранный сотрудник, то нельзя будет нажать кнопку. Является необязательным, и команда без него выглядела бы так:



Теперь надо связать окно с этим классом

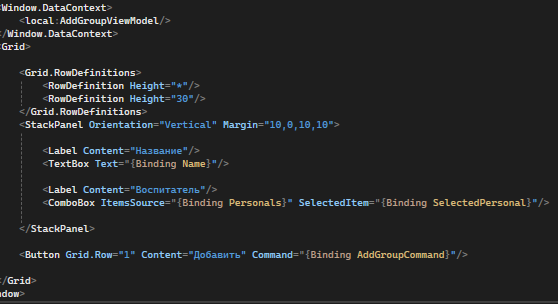


xmlns:local="clr-namespace:KidsKamp.ViewModel.AddViewModel" тут надо указать namespace файла



<local:AddGroupViewModel/> - название файла

После этого можно привязать свойства к элементам



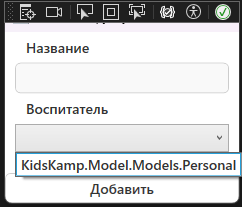
**Text="{Binding Name}"** – при изменении текста в Textbox у свойства Name будет меняться значение

**ItemsSource="{Binding Personals}"** – список воспитателей будет браться из списка Personals

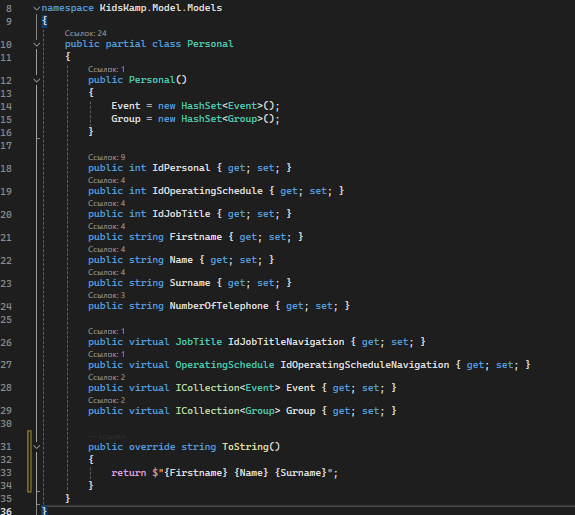
**SelectedItem="{Binding SelectedPersonal}"** – при выборе воспитателя значения **SelectedPersonal** в классе будет меняться

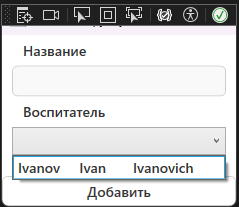
**Command="{Binding AddGroupCommand}** – при нажатии на кнопку будет выполняться команда **AddGroupCommand**

Вручную добавим в бд нескольких сотрудников, сделаем это окно стартовым и проверим работоспособность



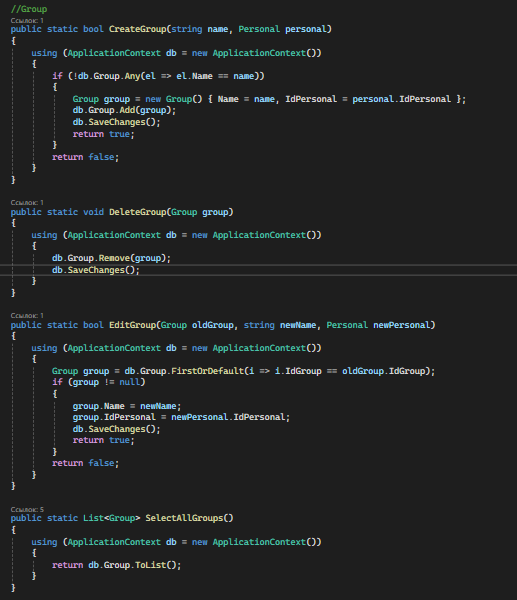
Работает, но список сотрудников отображается странно, что бы поменять это переходим в класс сотрудников и пишем следующий метод





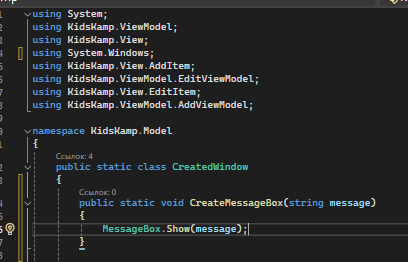
Теперь все работает, но не понятно создалась ли группа или нет, можно добавить всплывающее окно с результатом

Для этого нужно изменить метод добавления группы следующим образом и еще заодно можно поменять методы редактирования



Я поставил возвращаемое значение типа bool и при удачном добавлении/редактировании возвращается true

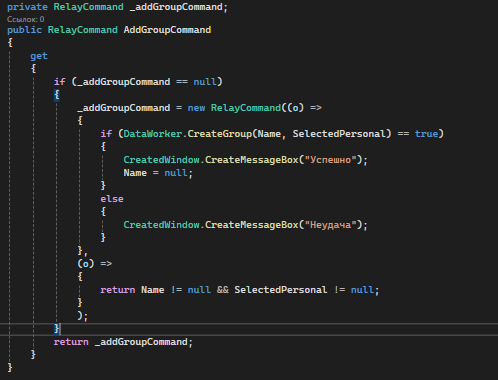
Создадим класс, который будет создавать окна с сообщением



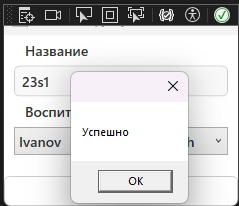
Важно добавить эту строку - **using System.Windows;**

В него надо будет передавать сообщении и будет выводиться окно

Теперь надо изменить команду в класс AddGroupViewModel

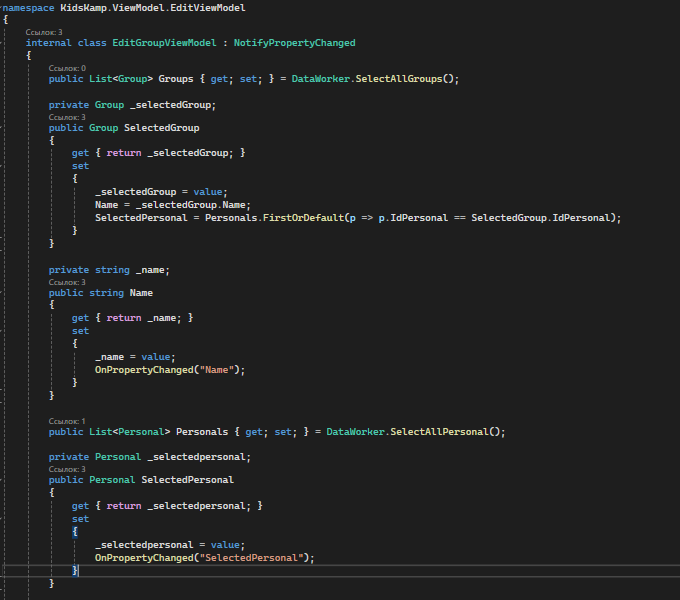


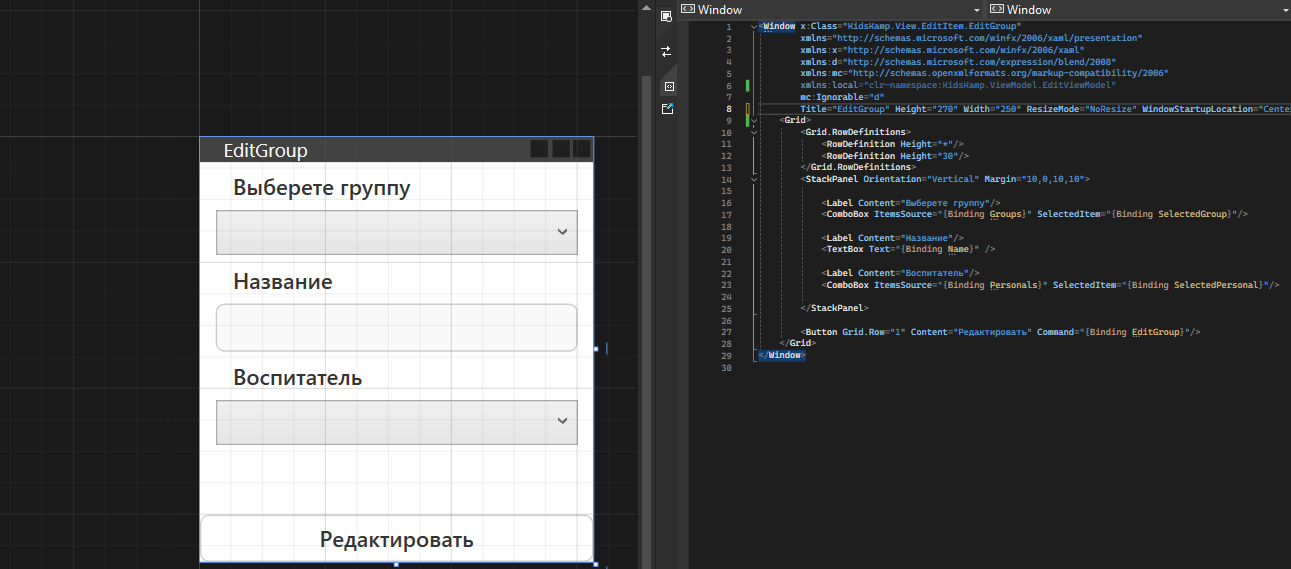
Проверим

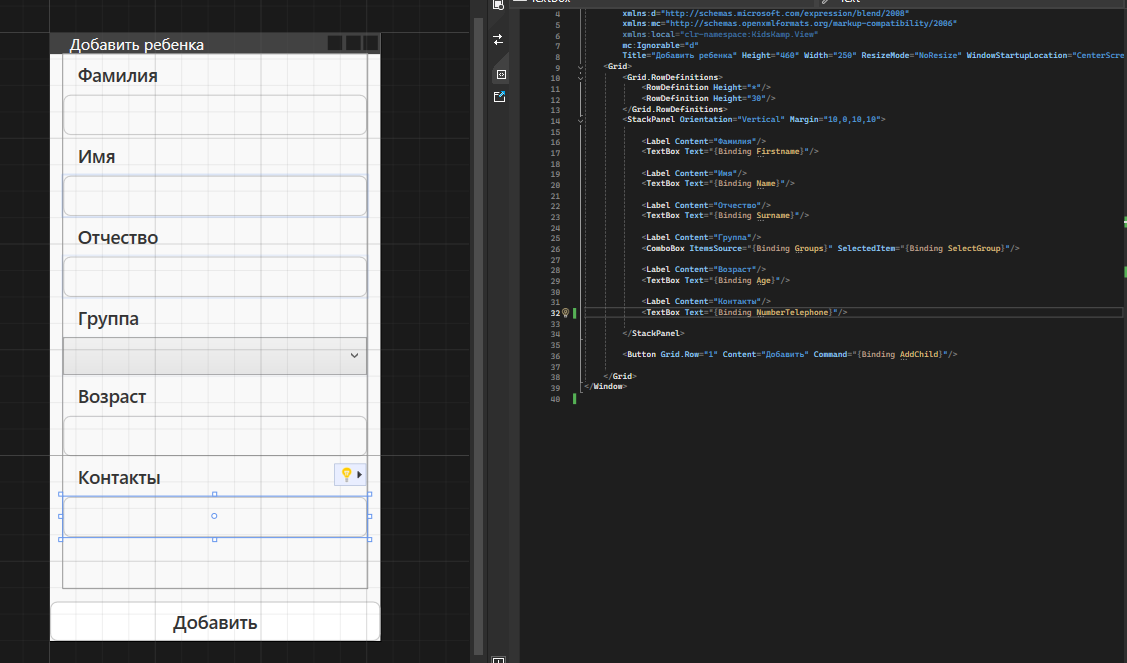


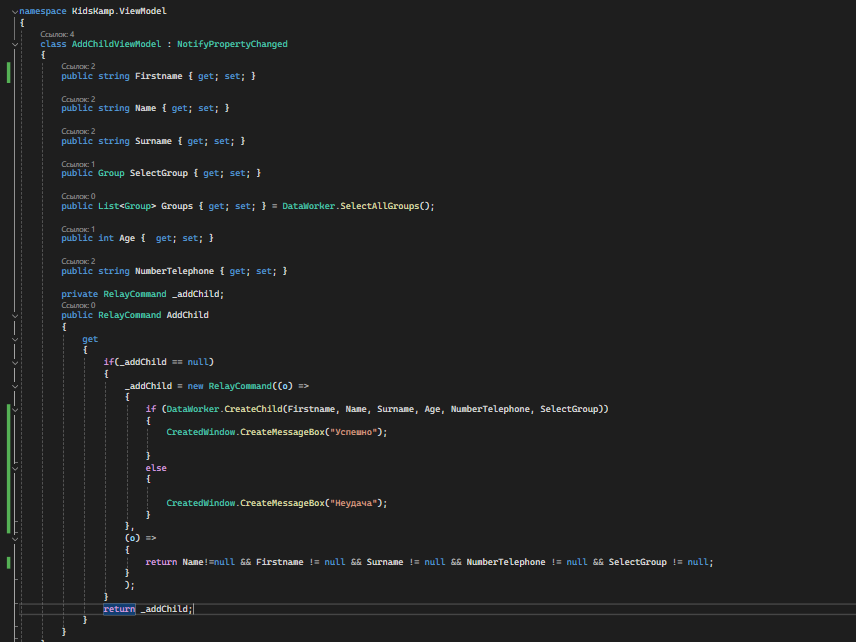
Теперь перейдем к окну **EditGroup**

Создадим класс **EditGroup**, в нем ничего нет принципиального нового, поэтому просто покажу его код

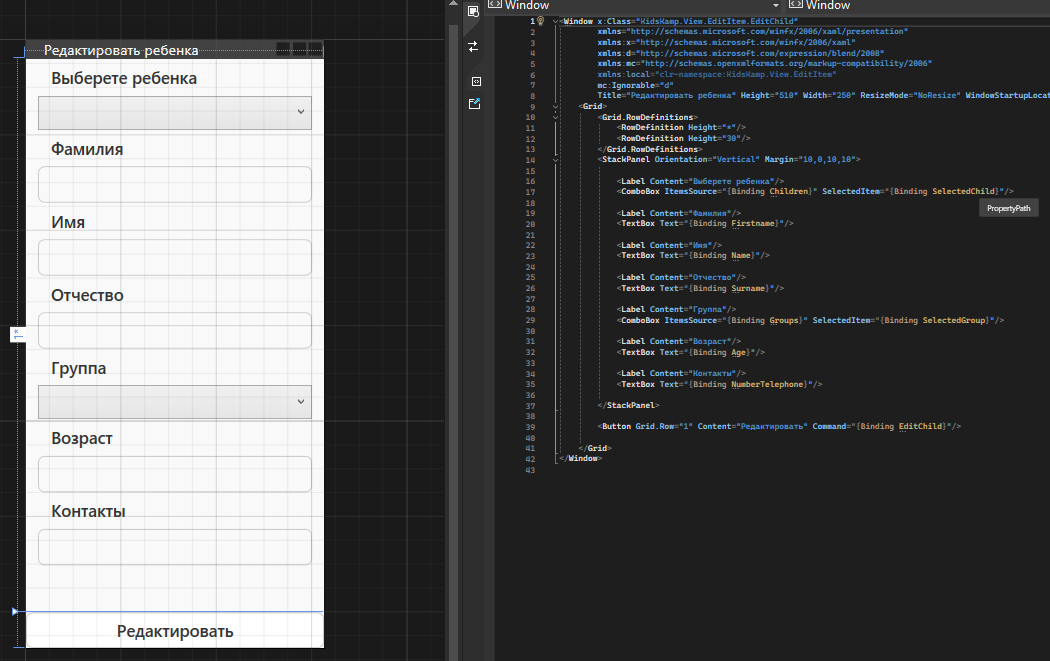


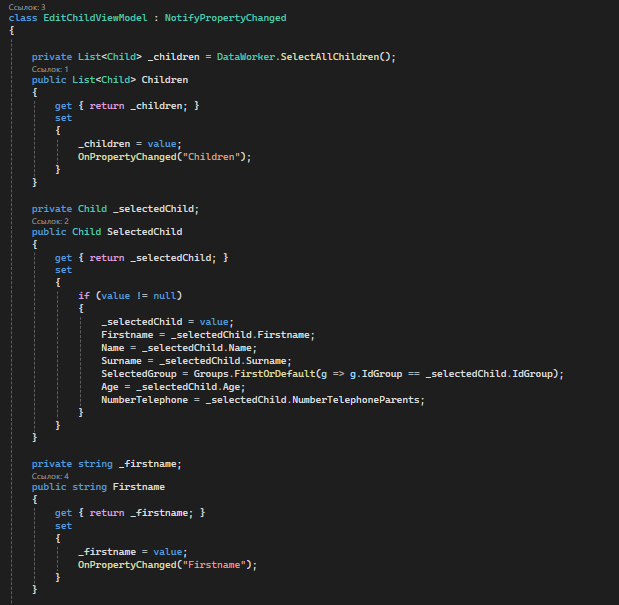


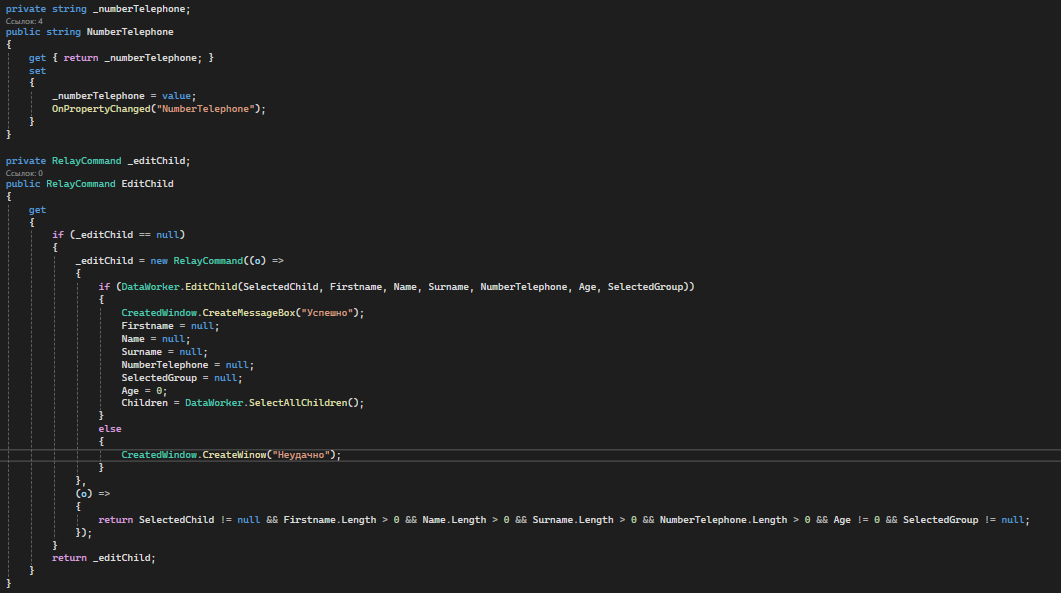




Окно редактирования ребенка





Теперь приступим к главному окну

Создадим класс **MainViewModel**

Реализуем открытие окон при нажатии на кнопки. Для этого в классе CreatedWindow создадим метод открытия окон.



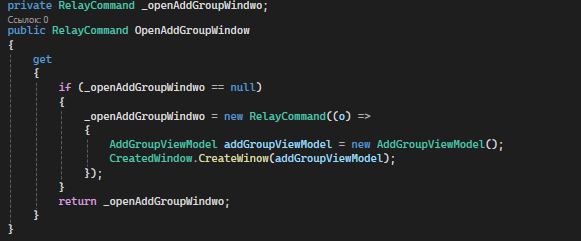
**dataContext is AddChildViewModel addChildView** – проверяет является ли **dataContext** классом **AddChildViewModel** и если да, то создает **addChildView**

**addChild.DataContext = addChildView;** - свзывает окно с ViewModel

**window.Visibility = Visibility.Visible;** - делает окно видимым

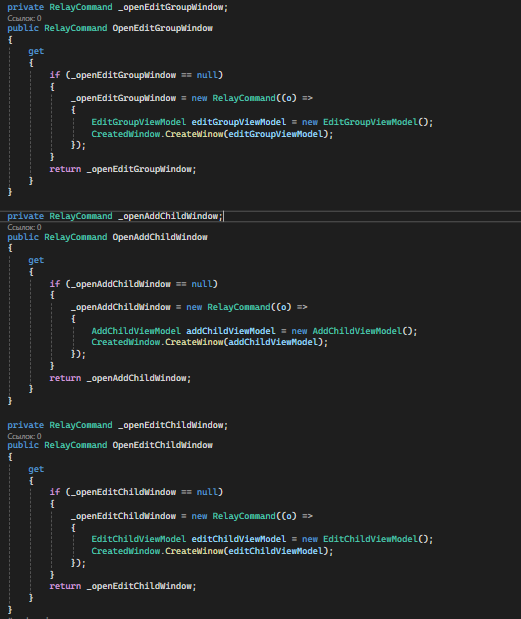
**window.Owner = Application.Current.MainWindow;** - делает дочерним окном главного

**window.ShowDialog();** - не дает использовать главное окно пока открыто дочернее



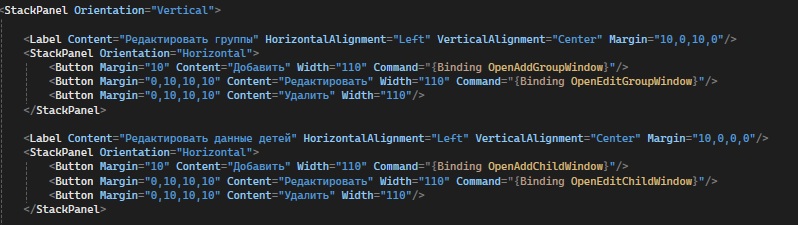
**AddGroupViewModel addGroupViewModel = new AddGroupViewModel();** - создаем **viewModel** окна добавления детей

Остальные команды выглядят аналогично



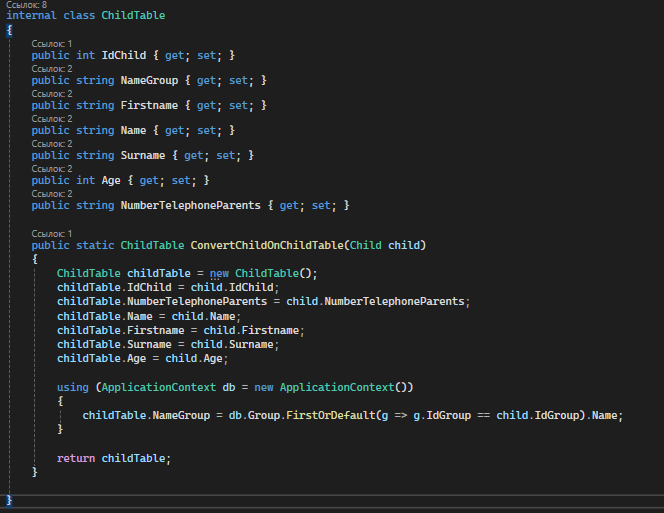
Команды для удаления реализуем позже

Теперь привяжем команды к кнопкам

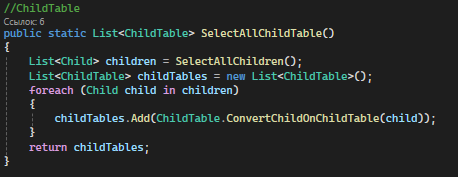


Теперь сделаем привязку таблицы

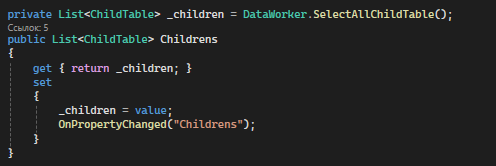
В классе **Child** хранится id группы, но нам нужно вывести ее название, поэтому нам надо создать класс который будет будет имя группы и надо создать метод конвертации обычного ребенка в ребенка для таблицы



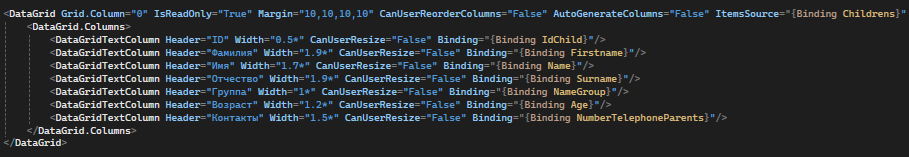
Теперь в классе **Dataworker** нужно создать метод отображения всех детей для таблицы



Теперь создадим в классе **MainViewModel** список со всеми детьми для таблицы

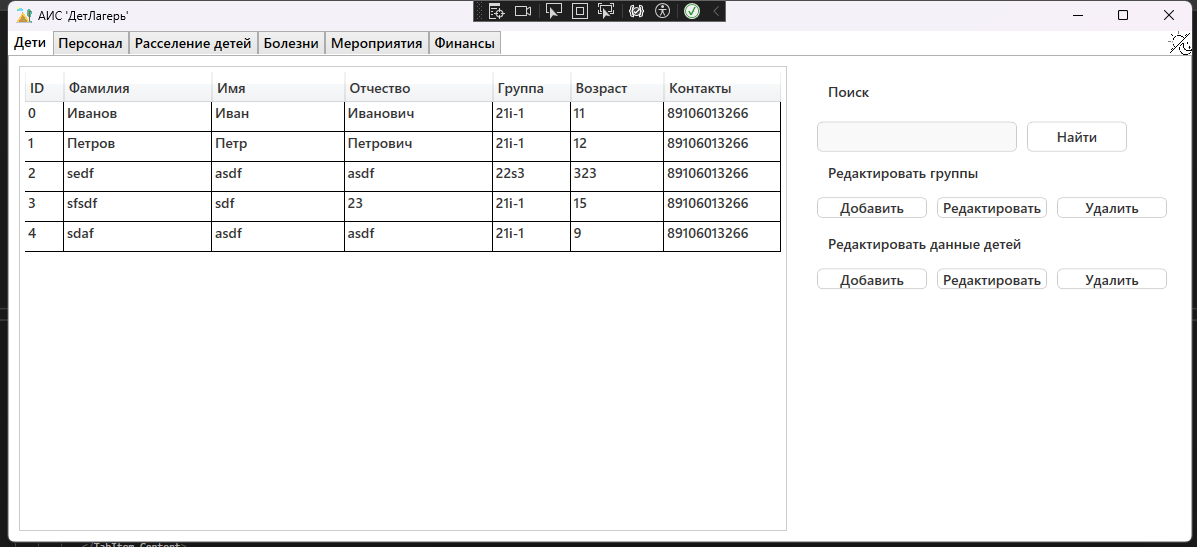


И привяжем этот список к таблице

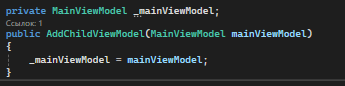


**ItetSource** - связывает таблицу со списком

**Binding="{Binding IdChild}** – связывает столбец таблицы с определенный свойством класса, который используется в качестве источника

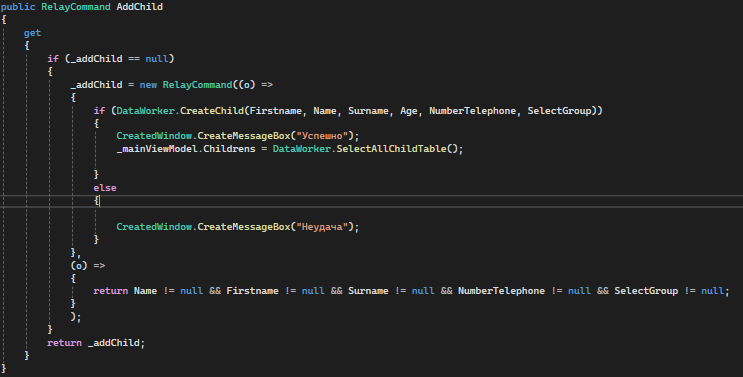


Все работает, но возникает проблема в том что при добавлени/изменние детей эти изменения не отображаются в таблице, чтобы это исправить изменяем класс **AddChildViewModel**



Теперь этот класс хранит в себе главное окно и запрашвает его при создании

И изменяем команду

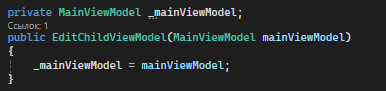


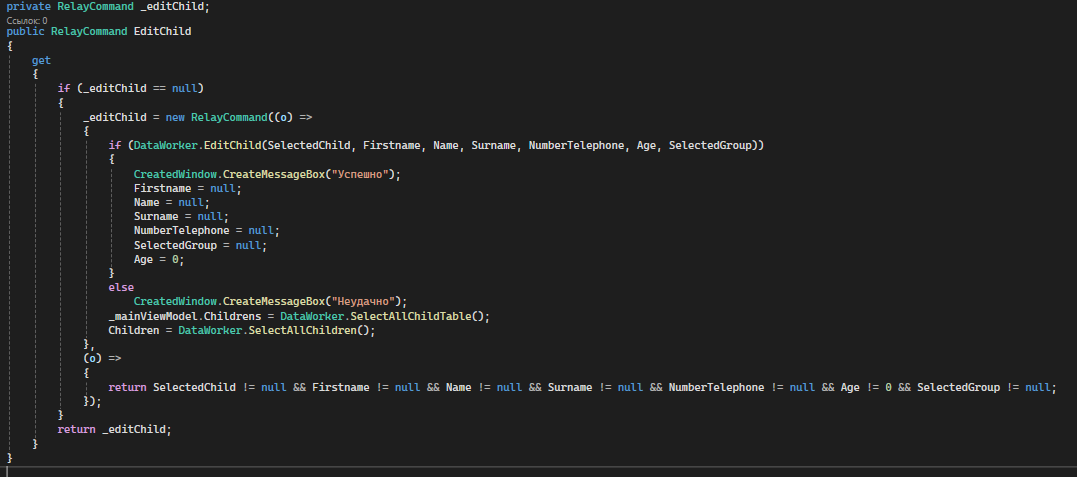
И в класс **MainWindowViewModel** надо передать параметр при создании окна

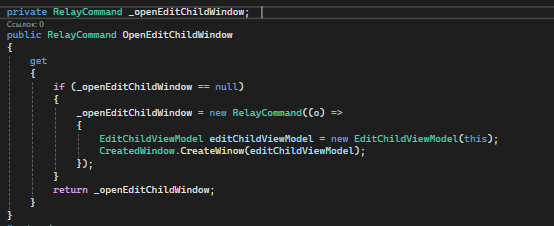


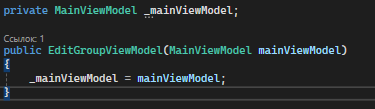
**this** – передает сам класс в качестве аргумента

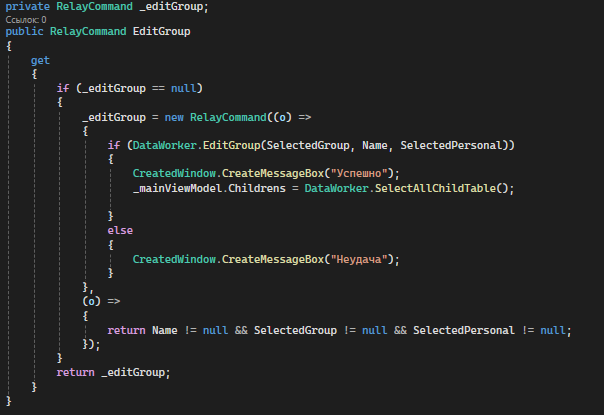
И теперь при успешном добавлении главное окно будет обновлять список детей.Аналогично надо сделать и при изменение данных ребенка и при изменение групп

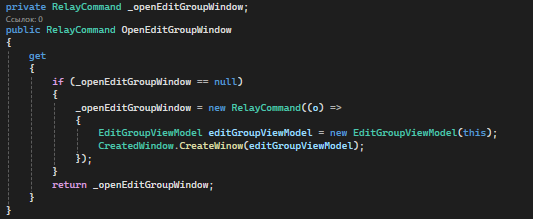




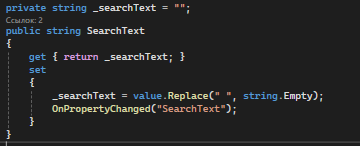








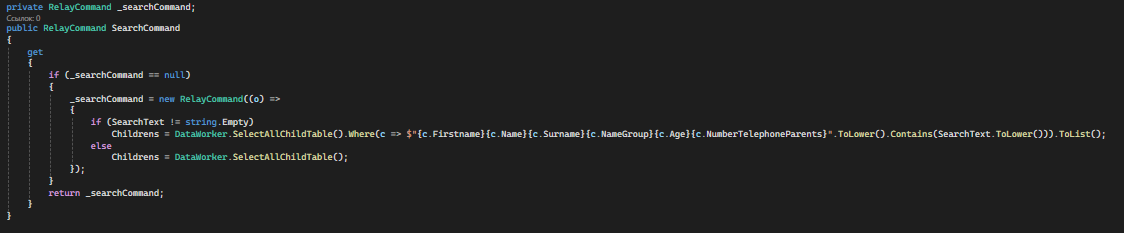
Теперь реализуем строку поиска для этого в классе **MainViewModel** создадим свойство



Метод **Replace** заменят символы которые передаются в первом параметре на символы которые передаются вторым параметром

**String.Empty** – пустая строка, то есть “”.

И создадим команду поиска



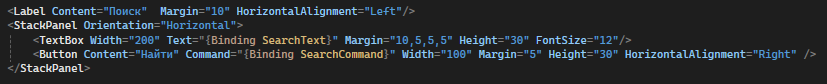
Проверяем не равна ли строка поиска пустоте и если нет, то ищем, а если да, то показываем всех детей

**Where** – ищет детей по указанному условию

**ToLower** – преобразует все символы в строке к нижнему регистру

**Contains** - проверяет содержится ли одна строка в другой

И привязываем свойство с командой



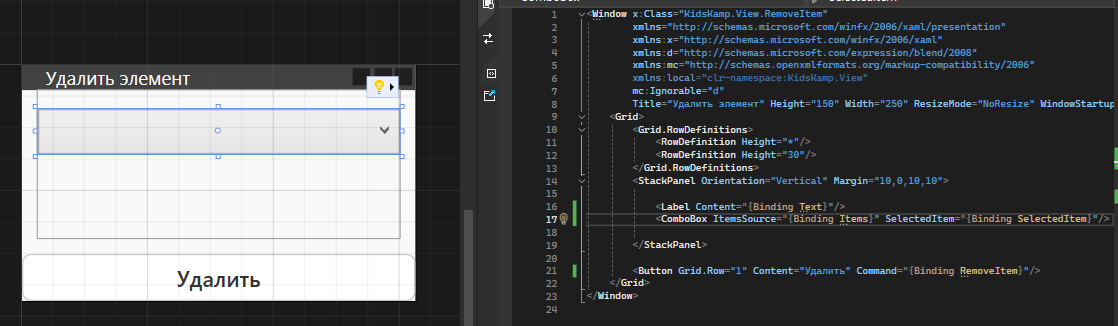
Реализуем окно удаления





Тут используется Try Catch, он перехватывает ошибку и обрабатывает ее, ошибка может произойти из-за того, что элемент таблицы используется в другой таблице и его нельзя удалить. Подробнее про Try Cacth:

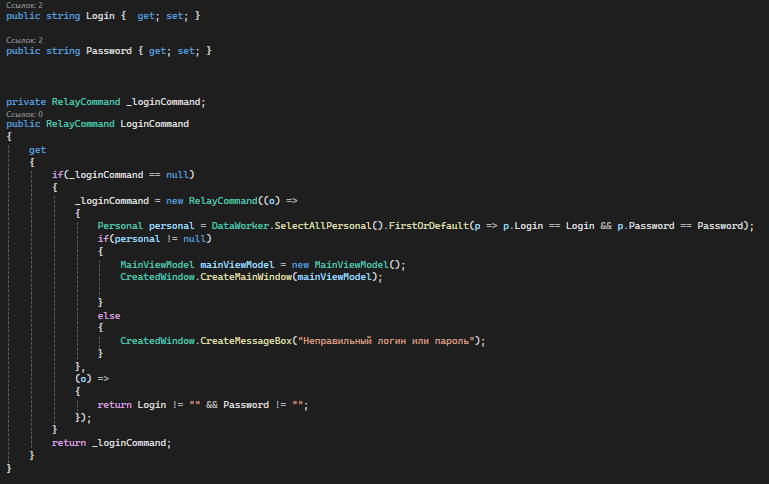
* https://metanit.com/sharp/tutorial/2.14.php

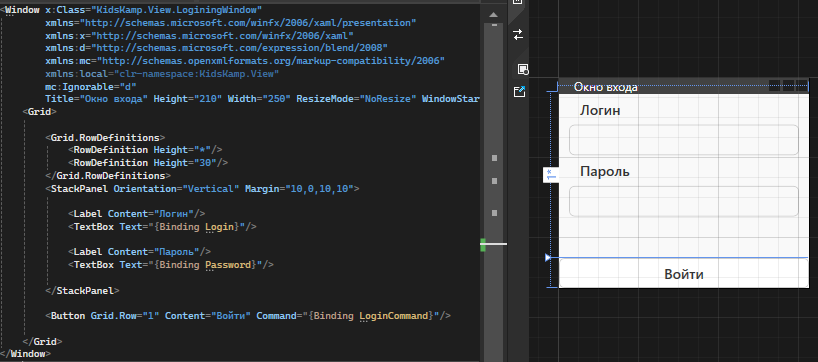


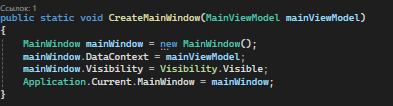
И теперь реализуем команды открытия окон удаления.



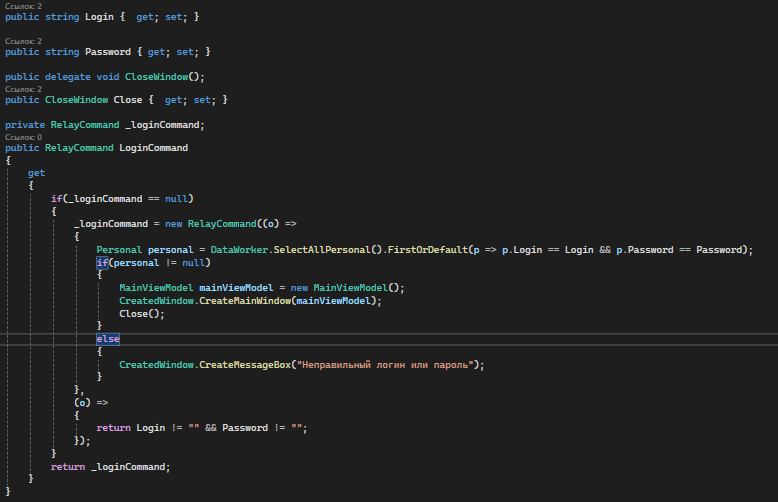
Теперь реализуем окно входа в программу. Создаем класс LoginingViewModel

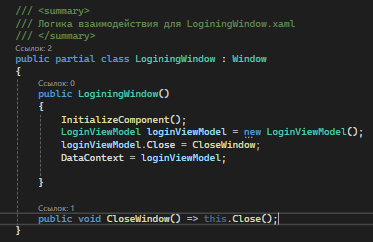




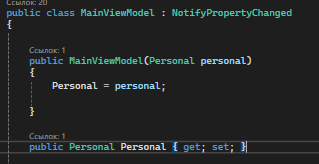


Чтобы реализовать закрытие окна входа при открытие главного окна, надо использовать делегат



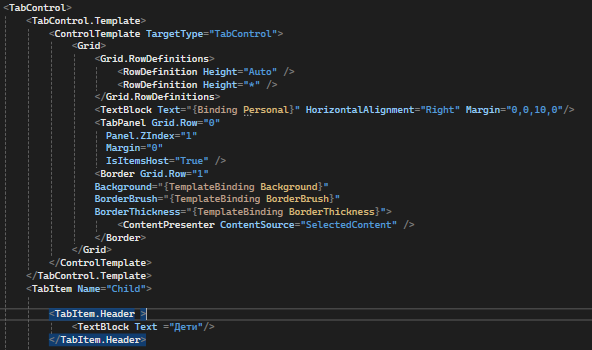


Сделаем чтобы на главном окне отображалось Фио, вошедшего, сотрудника

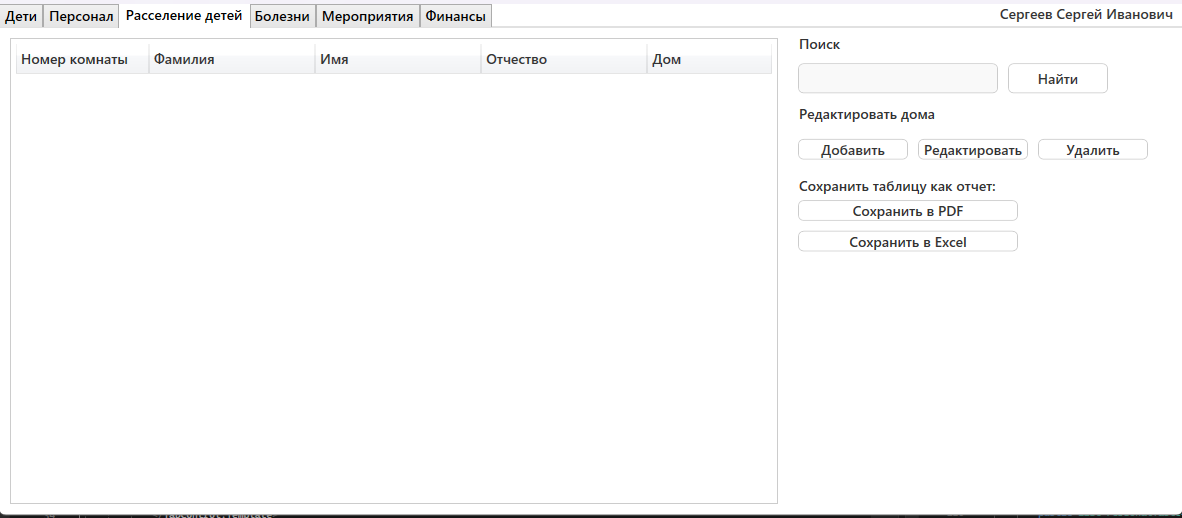




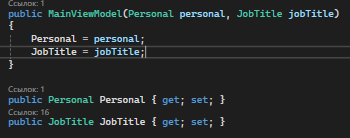
И привяжем к окну

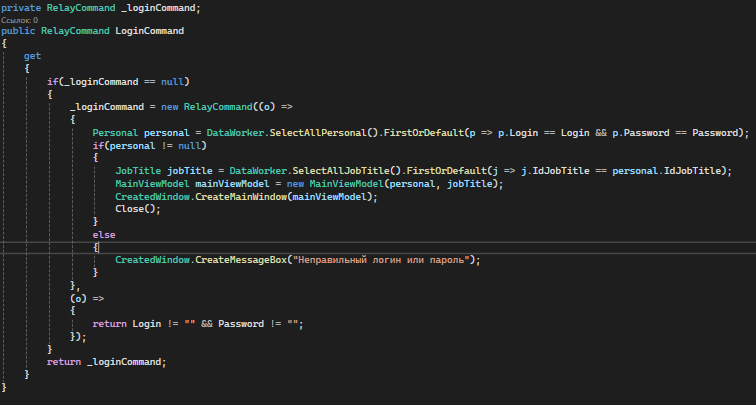


Выглядеть это будет так

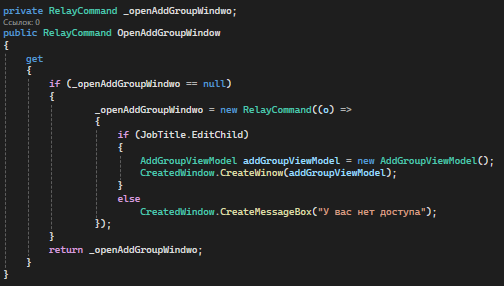


Теперь реализуем разные права для должностей для этого нам надо знать должность сотрудника



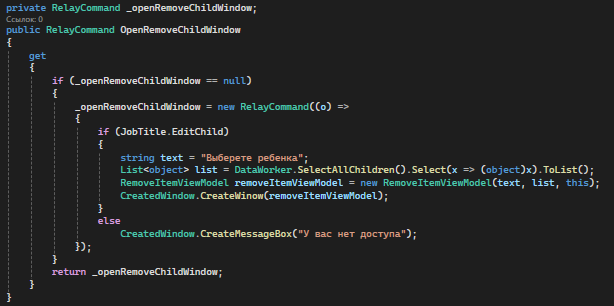


И теперь надо отредактировать команды открытия окон







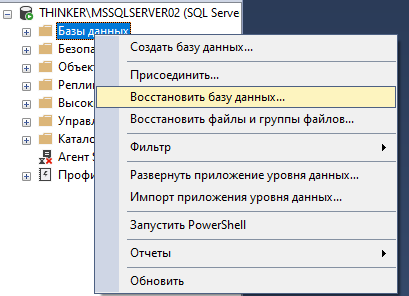


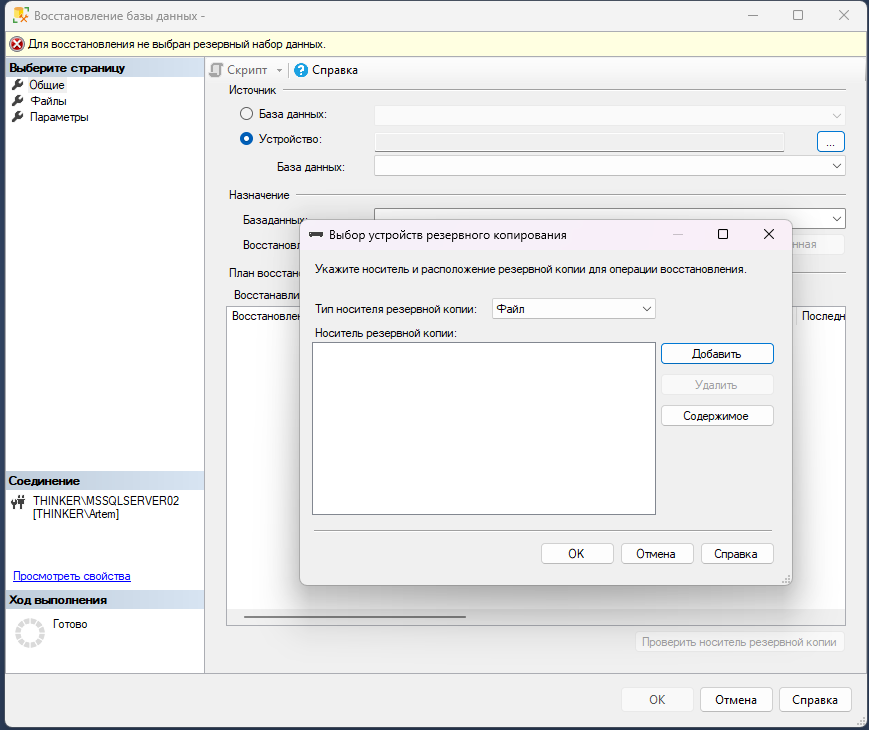
Если у вошедшего сотрудника в должности разрешено редактирование детей, то ему будет открываться окно добавления группы, а если нет, то будет выводится сообщение о том, что у него нет доступа. С другими командами аналогично

Вот пример работы

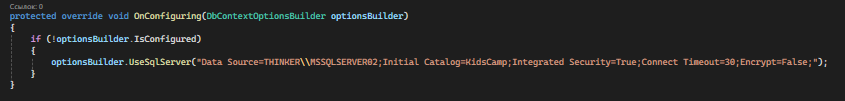
https://drive.google.com/drive/folders/1AK1BNaQB-C6azb7KKULvLpbUhlVw2ZKE?usp=sharing

Чтобы импортировать бд





И в ApplicationContext поменять имя сервера



# **Функциональное тестирование**

Это самый популярный вид тестирования, который проверяет соответствие функциональности продукта тому, как он был задуман.

С помощью функционального тестирования можно проверить:

● работоспособность приложения или сайта;

● его основные функции.

Что такое Тест Сьют и для чего он нужен?

Тест Сьют это набор [тест кейсов](http://software-testing.org/testing/chto-takoe-test-keys-test-case-iz-kakih-poley-sostoit-tipichnyy-test-keys.html), которые объединены тем что относятся к одному тестируемому модулю, функциональности, приоритету или одному типу тестирования. Каждый тест сьют состоит из более чем одного тест кейса и зачастую выполняется всей «пачкой» в процессе тестирования.

Примеры оформления списка тест-сьютов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Автор | Приоритет | Заголовок | Список тест-кейсов |
| 1 | user | 1 | Модуль пользователя | * 1. Авторизация   2. Регистрация   3. Редактирование профиля |
| 2 | user | 1 | Модуль товаров | * 1. Получение новинок   2. Получение товаров в категории   3. Получение товара |
| 3 | user | 1 | Модуль корзины | * 1. Добавление товара в корзину   2. Увеличение количества товара в корзине   3. Уменьшение количества товара в корзине |
| 4 | user | 1 | Модуль оформления заказа | * 1. Оформление заказа   2. Оформление заказа   3. Оформление заказа |

Тест-кейсы объединяют в тест-сьюты для большего удобства при прохождении тест кейсов, проходя их последовательно от модуля к модулю, от одного типа тестирования к другому а не сумбурно, бросаясь из одного угла в угол, оставив не проверенным большую часть модуля или общей функциональности.

Что такое тест-кейсы?

Тест-кейс — это алгоритм действий, которые требуется совершить для проверки работы программы (кнопок, полей ввода и т.д.). В него входят шаги, которые предпринимаются перед проверкой (предусловия), являются проверкой, а также ожидаемый результат — то, что получим после выполненных действий.

Примеры оформления списка тест-кейсов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Описание  (Тип) | Предусловия | Шаги | Ожидаемый  результат |
| 1.1 | Авторизация (позитивный) | Пользователь находится на странице входа в личный кабинет  Пользователь был ранее зарегистрирован в системе | Ввести в поля «Логин» и  «Пароль» логин и пароль пользователя  Нажать кнопку «Войти» | 1. Пользователь перенаправляется на страницу личного кабинета |
| 1.2 | Регистрация (негативный) | Пользователь находится на странице регистрации  Пользователь был ранее зарегистрирован в системе | Ввести в обязательные поля «Имя», «Фамилия»,  «Пароль», «Подтвердите пароль», «Электронная почта» данные. Также можно ввести данные в поля  «Номер телефона» и  «Адрес».  Нажать кнопку  «Зарегистрироваться» | 1. Пользователь на странице регистрации получает сообщение  «Пользователь с таким email существует» |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Описание (Тип) | Предусловия | Шаги | Ожидаемый результат |
| 1.3 | Редактирование профиля (негативный) | 1. Пользователь находится на странице личного кабинета | Нажать на пункт  «Профиль»  Получить страницу редактирования профиля  Нажать на кнопку  «Изменить»  Ввести в поля  «Пароль» и  «Подтвердите пароль» разные значения  Нажать на кнопку  «Сохранить» | 1. Пользователю отображается сообщение «Пароли не совпадают» |
| 2.1 | Получение новинок (Позитивный) | 1. Пользователь находится на одной из страниц магазина | 1. Пользователь нажимает на название интернет-магазина, расположенного в  шапке сайта | 1. Пользователю отображается главная страница магазина со списком новинок |
| 2.2 | Получение товаров в категории (Позитивный) | 1. Пользователь находится на  одной из  следующих страниц: на главной странице, на странице  «Корзина», на странице входа в личный кабинет, на странице регистрации, на странице каталога  товаров, на странице товара | 1. Нажать на пункт меню категорий | Пользователь перенаправляется на страницу категорий  На странице категорий отображается заголовок, совпадающий с названием ранее выбранного пункта меню |
| 2.3 | Получение товара (Позитивный) | Пользователь находится на  одной из  следующих страниц: на главной странице, на странице  «Корзина», на странице категорий  На данной странице  отображается товар | 1. Нажать на кнопку  «Товар» | Пользователь перенаправляется на страницу товара  Информация о товаре на странице товара должна совпадать с  информацией на странице, с которой был осуществлен переход |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Описани  е (Тип) | Шаги | Ожидаемый  результат | Ожидаемый  результат |
| 3.1 | Добавле ние товара в корзину (Позитив ный) | Пользователь находится на одной из следующих страниц: на главной странице, на странице категорий, на странице товара  На данной странице отображается товар | 1. Нажать на кнопку «Добавить в корзину» | Страница, на которой находится пользователь, обновляется  Пользователю отображается сообщение «Товар в корзину успешно добавлен»  При переходе на страницу «Корзина» отображается добавленный товар |
| 3.2 | Увеличен ие количест ва товара в корзине (Позитив  ный) | 1. Пользователь находится на странице «Корзина» | 1. Нажать на кнопку «+» | 1. В колонке «Кол- во» в строке товара увеличивается значение на 1 |
| 3.3 | Уменьше ние количест ва товара в корзине (Негатив  ный) | Пользователь находится на странице «Корзина»  Значение в колонке «Кол- во» в строке товара равно 1 | 1. Нажать на кнопку «-» | 1. Значение в колонке  «Кол-во» в строке товара не изменяется |