**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Специальность

1-40 04 01 «Информатика и технологии программирования»

Кафедра «Информатика»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

по дисциплине «ООП»

на тему: **Разработка информационной системы**

**«СТРАХОВоЕ АГЕНСТВО»**

Исполнитель: студент гр. ИП-22

Э. С. Суховенко

Руководитель: ст. преподаватель Я. М. Олизарович

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: ­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсовой работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2021

**СОДЕРЖАНИЕ**

|  |  |  |  |
| --- | --- | --- | --- |
| Введение | | | 3 |
| 1. Аналитический обзор | | | 4 |
|  | 1.1 Описание объекта | | 4 |
|  | 1.2 Обзор существующих методов | | 4 |
|  | 1.3 Требования к проектируемому программному обеспечению | | 5 |
| 2. Архитектура программного обеспечения | | | 6 |
|  | 2.1 О модели предметной области | | 6 |
|  | 2.2 Преценденты и актёры | | 6 |
| 3.Структура программного обеспечения | | | 9 |
|  | 3.1 Используемые средства | | 9 |
|  | 3.2 Модель данных | | 10 |
|  | 3.3 Доменные классы | | 11 |
|  | 3.4 Реализация интерфейса | | 15 |
| 4. Тестирование | | | 21 |
|  | | 4.1 Верификация | 21 |
|  | | 4.2 Обработка исключительных ситуаций | 21 |
| Заключение | | | 25 |
| Список использованных источников | | | 26 |
| Приложение А – Листинг программы | | | 27 |

**ВВЕДЕНИЕ**

Широкое использование средств вычислительной техники привело к появлению новых приемов и способов организации учета. Учёт информации о поступающих заказов является самым сложным и трудоемким процессом учета, поэтому использование компьютерных технологий при обработке информации просто необходимо.

Автоматизация какого-либо объекта подразумевает введение в него функций, выполняемых машиной, а не человеком, это позволяет ускорить процесс введения и обработки данных, тем самым повысив эффективность организации.

Страхование ­- необходимый элемент производственных отношений. Оно связано с возмещением материальных потерь в процессе общественного производства. Рисковый характер общественного производства, порождает отношения между людьми по предупреждению, преодолению, локализации и по безусловному возмещению нанесенного ущерба.

Актуальность рассматриваемого вопроса усиливается еще и потому, что в современном обществе, наряду с традиционным предназначением - обеспечением защиты от природной стихии (землетрясения, наводнения, бури и др.), случайных событий технического и технологического характера (пожары, аварии, взрывы и др.), - объектом страхования все больше становятся убытки от различных криминогенных явлений (кражи, разбойные нападения, угон транспортных средств и др.)

Целью данной курсовой работы является автоматизация деятельности страхового агенства. В ходе выполнения курсовой работы будет разработано программное обеспечение, способное автоматизировать работу страхового агенства.

**1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

**1.1 Описание объекта**

Страховая компания — это исторически определенная общественная форма функционирования страхового фонда, обособленная структура, осуществляющая заключение договоров страхования и их обслуживание. Страховой компании свойственны технико-организационное единство и обособленность. Экономическая обособленность страховой компании проявляется в полной обособленности его ресурсов, их полном самостоятельном обороте. Страховая компания функционирует в экономической системе в качестве самостоятельного хозяйствующего субъекта и "встроена" в определенную систему производственных отношений. Экономически обособленные страховые компании строят свои отношения с другими страховщиками на основе перестрахования и сострахования.

Однако часто в подобных компаниях возникают подобные или полностью совпадающие операции, например, оформление договра, регистрация клиента, оплата договора и т. п. Эти операции можно и нужно автоматизировать.

Разработанная система автоматизации должна вести учёт вышеперечисленного, позволять клиенту выбрать интересующие его услуги, добавить их в корзину и оформить заказ. Это существенно упростит работу агента. В свою очередь менеджер с помощью разработанной системы всегда сможет получить последние актуальные сведения, связанные с работой страхового агенства.

Итогом проектирования будет настольное приложение под управлением операционной системы Windows с подключенной базой данных, в которой будет хранится информация о пользователях, их ролях, заказах клиентов, доступных услугах.

**1.2 Обзор существующих методов и средств решения**

Основная задача разрабатываемой программы – оказание услуг по страхованию физических и юридических лиц. Если страховое агенство имеет достаточно большой поток клиентов, то возникает необходимость автоматизировать его деятельность. В первую очередь рост потребности в страховых услугах связан с постоянным увеличением количества несчастных случаев на производстве. Автоматизация позволяет уделить больше времени на оказание услуг, сократить время их выполнения, и, как следствие, увеличить эффективность работы компании.

В ходе изучения аналогов приложений для автоматизации заданной предметной области был рассмотрен сервис для автоматизации страховых услуг “СтрахОнлайн”. Его основными преимуществами являются: возможность автоматизации нескольких предметных областей, планирование загрузки мастеров в департаменте, учёт клиентов компании, планирование загрузки мастеров, наличие приложений для мобильных платформ. Ведение склада запчастей.

Основными преимуществами проектируемой программы является простота использования, отсутствие необходимости оформления подписки на использование, наличие приложения для OC Windows.

В заключении, изучив функционал сервиса “СтрахОнлайн” и других аналогов, можно сделать вывод, что основными требованиями для проектируемой предметной области являются: хранение базы пользователей, сбор статистики о работе агенства, продвижение заказа.

**1.3 Требования к проектируемому программному обеспечению**

В качестве предметной области автоматизации рассматривается автоматизация деятельности страхового агенства. Проектируемое программное обеспечение должно быть удобным для обработки данных, связанных с оказанием услуг по различным видам страхования.

Автоматизация работы страхового агенства должна позволить рабочим проще и быстрее узнавать информацию по поступившим заказам, отмечать выполнение заказа и завершать его, а также вести справочник, который должен состоять из таких пунктов, как: категории услуг, виды страхования, услуги, и, наконец, прейскурант, который устанавливает соответствие между видом страхования и услугой, добавляя при этом к этой паре описание и цену.

Проектируемое программное обеспечение должно уметь упорядочить огромный объём данных, поступающий в ходе работы компании, в презентабельном для руководителя виде, создавая отчёты, к которым относятся отчёт по прейскуранту и по объёму оказанных агенством услуг за один календарный месяц. Также оно должно уметь разграничивать возможности работников, вводя для этого 3 роли. К ним относятся роль покупателя, агента, менеджера. Возможности роли покупателя были описаны выше. В полномочия роли агента должно входить согласование заказов и их подтверждение. В полномочия роли менеджера должно входить отслеживание продуктивности стрховых агентов и департамента, к которому он прикреплён, а также введение поощрений для тех сотрудников, которые показали себя особенно хорошо.

Подводя итог в этом разделе можно сказать, что все требования к проектируемому программному обеспечению, описанные выше, должны максимально возможно автоматизировать страховое агенство, сведя к минимуму временные затраты, тем самым ускоряя обслуживание клиентов и увеличения качество их обслуживания. А предоставляемая менеджеру отчётность должна способствовать развитию агенства.

**2 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**2.1 О модели предметной области**

Главная цель построения логической модели – это получение графического представления логической структуры исследуемой предметной области. Логическая модель предметной области иллюстрирует сущности, а также их взаимоотношения между собой. Сущности описывают объекты, являющиеся предметом деятельности предметной области, и субъекты, осуществляющие деятельность в рамках предметной области. Взаимоотношения между сущностями иллюстрируются с помощью связей. Обычно связи определяют либо зависимости между сущностями, либо влияние одной сущности на другую. Важно понимать, какой именно информацией система должна управлять. Для этого нужно знать, какие объекты попадают в предметную область проектируемой ИС и какие логические связи между ними существуют.

Разработка логической модели должна выполняться последовательно, по мере проработки предметной области и поставленной задачи, уточняться и детализироваться.

Логическую модель надо строить так, чтобы сущности назывались именами существительными, связи глаголами, а чтение диаграммы рождало бы предложения, описывающие то, что происходит в предметной области.

По мере проработки модели уточняется состав сущностей и связей, а также определяются атрибуты сущностей.

Важно помнить, что логическая модель это не про структуру базы данных, это про логическую структуру предметной области задачи. Исключая ее из разрабатываемых атрибутов, мы лишаем себя эффективного инструмента анализа и проектирования, позволяющего очень точно учесть аспекты бизнеса, не иллюстрируемые динамическими моделями.

**2.2 Преценденты и актёры**

Для того, чтобы создать качественную систему автоматизации, недостаточно понять бизнес-процессы и потребности Заказчика. Важно понимать, какой именно информацией система должна управлять. А для этого нужно знать, какие объекты попадают в предметную область проектируемой ИС и какие логические связи между ними существуют. Для формирования такого понимания используются логические модели предметной области. При подготовке модели используем свободно распространяемый инструмент проектирования UML диаграмм – StarUML и методологию объектно-ориентированного анализа и проектирования.

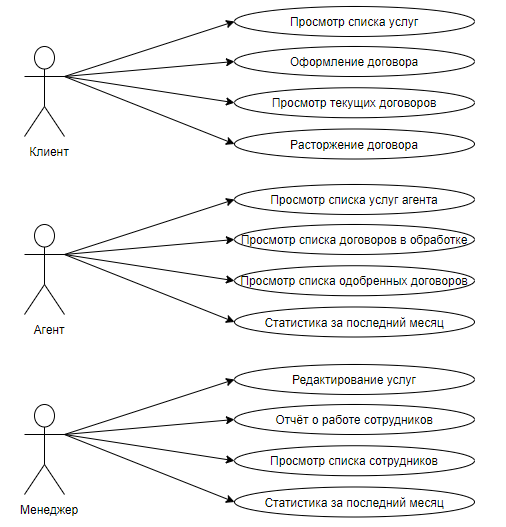


Рисунок 2.1 – Диаграмма прецендентов и актёров

Далее приведено описание актёров.

**Актёр “Клиент”** – это актёр, главная задача которого – формирование корзины услуг и последующее оформление заказа на их основе. Также в его полномочия входят: просмотр списка услуг, управление корзиной (добавление услуг, изменение их количества и удаление услуг), добавление отзыва на завершённый заказ.

**Актёр “Агент”** – актёр, главная задача которого оформление договора, он решает одобрить или отказать клиенту в страховании на основе определённых критериев.

**Актёр “Менеджер”** – это актёр, который может просматривать отчётность о работе агенства. Она состоит из отчётов, статистики и отзывов. Отчёты включают в себя сведения по услугам прейскуранта и по объёму оказанных услуг за прошлый месяц. Статистика включает в себя сведения за определённый период о новых пользователях и сведения о ежедневном количестве заказов за тот же период.

Далее приведено описание прецедентов.

**Прецендент “Просмотр списка услуг”** нужен для ознакомления с видами страхования и стоимостью их оформления.

**Прецендент “Оформление договора”** нужен для закрепления выбранной услуги и её последующей оплаты.

**Прецендент “Просмотр текущих договоров”** нужен для ознакомления клиента с его действующими договорами.

**Прецендент “Расторжение договора”** нужен для оформления отказа от уже оформленной услуги.

**Прецендент “Просмотр списка услуг агента”** нужен для ознакомления агента с его полномочиями.

**Прецендент “Просмотр списка договоров в обработке”** нужен для вынесения решения по договору.

**Прецендент “Просмотр списка одобренных договоров”** нужен для ознакомления агента с выполненной работой.

**Прецендент “Статистика за последний месяц”** нужен для проверки заработной платы.

**Прецендент “Редактирование услуг”** нужен для внесения изменений в список услуг.

**Прецендент “Отчёт о работе сотрудников”** нужен для оценки продуктивности агентов.

**Прецендент “Просмотр списка сотрудников”** нужен для поощрения или наказания сотрудников по результатам их работы.

**Прецендент “Премирование сотрудника”** нужен для поощрения сотрудников по результатам их работы.

**Прецендент “Депремирование сотрудника”** нужен для наказания сотрудников по результатам их работы.

**Прецендент “Добавление услуги”** нужен для добавления информации о новой услуге. Используется при расширении компании.

**Прецендент “Редактирование услуги”** нужен для изменения информации об услуге, например, можно изменить цену или описание.

**Прецендент “Удаление услуги”** нужен для удаления услуги, которая больше не поддерживается агенством.

**3 СТРУКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**3.1 Используемые средства**

Для создания системы автоматизации был выбран объектно-ориентированный язык программирования C#, который является языком разработки приложений для платформы Microsoft .NET Framework и основой для использования технологии WPF. Данная технология имеет удобный инструментарий для разработки приложений для рабочего стола Windows.

Название «Си шарп» (от англ. Sharp — диез) происходит от буквенной музыкальной нотации, где букве C соответствует нота До, а знак диез означает повышение соответствующего ноте звука на полутон. Название также является игрой с цепочкой C → C++ → C++++(C#), так как символ «#» можно представить состоящим из 4 знаков «+». Он относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Delphi, и, в особенности, Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов.

Для хранения и удобного просмотра данных используется база данных MS Access.

Для управления записями в таблице БД был выбран ADO.NET. Он предоставляет собой технологию работы с данными, которая основана на платформе .NET Framework. Эта технология представляет нам набор классов, через которые мы можем отправлять запросы к базам данных, устанавливать подключения, получать ответ от базы данных и производить ряд других операций.

Причем важно отметить, что систем управления баз данных может быть множество. В своей сущности они могут различаться. MS SQL Server, например, для создания запросов использует язык T-SQL, а MySQL и Oracle применяют язык PL-SQL. Разные системы баз данных могут иметь разные типы данных. Также могут различаться какие-то другие моменты. Однако функционал ADO.NET построен таким образом, чтобы предоставить разработчикам унифицированный интерфейс для работы с самыми различными СУБД.

Основу интерфейса взаимодействия с базами данных в ADO.NET представляет ограниченный круг объектов: Connection, Command, DataReader, DataSet и DataAdapter. С помощью объекта **Connection** происходит установка подключения к источнику данных. Объект **Command** позволяет выполнять операции с данными из БД. Объект **DataReader** считывает полученные в результате запроса данные. Объект **DataSet** предназначен для хранения данных из БД и позволяет работать с ними независимо от БД. И объект **DataAdapter** является посредником между DataSet и источником данных. Главным образом, через эти объекты и будет идти работа с базой данных.

Однако чтобы использовать один и тот же набор объектов для разных источников данных, необходим соответствующий **провайдер данных**. Собственно через провайдер данных в ADO.NET и осуществляется взаимодействие с базой данных. Причем для каждого источника данных в ADO.NET может быть свой провайдер, который собственно и определяет конкретную реализацию вышеуказанных классов.

При работе с ADO.NET был выбран провайдер OleDb.

Для связи сущностей базы данных с объектами C# была реализована так называемая “анемичная модель”, то есть каждой сущности БД был поставлен в соответствие объект C# без методов.

**3.2 Модель данных**

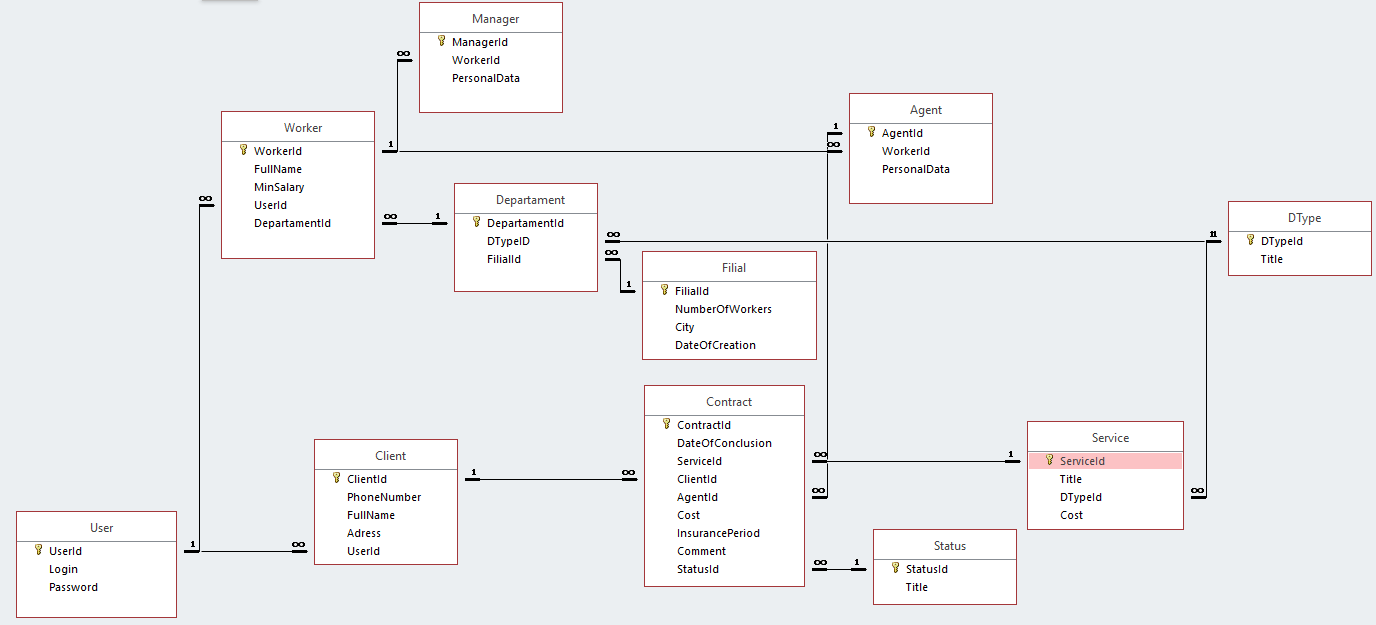


Рисунок 3.1 – Модель данных

Рассмотрим подробно назначения таблиц модели данных, показанных на рисунке 3.1.

Таблица “User” предназначена для хранения авторизационных данных учётной записи, а именно логина и пароля.

Таблица “Worker” предназначена для хранения базовой информации о работнике компании, вне зависимости от должности, например, ФИО, адрес проживания, номер телефона и т.п.

Таблица “Client” предназначена для хранения информации о клиенте.

Таблица “Agent” предназначена для хранения информации о агенте, здесь хранится более детальная информация относительно должности, например, навыки агента, его стаж и опыт.

Таблица “Manager” предназначена для хранения информации о менеджере, здесь хранится более детальная информация относительно должности, например, предыдущее место работы.

Таблица “Departament” предназначена для хранения информации о департаменте, эта таблица связывает департамент и филиал.

Таблица “Filial” предназначена для хранения информации о филиале, например, местоположение филиала, количество сотрудников, дата открытия, состояние на данный момент.

Таблица “Contract” предназначена для хранения информации о договорах, а именно коды клиента и агента, участвовавших в оформлении договора, итоговая стоимость, период страхования, статус.

Таблица “DType” предназначена для хранения информации о типах департаментов, эта таблица задаёт общие департаменты для разных филиалов, таким образом вне зависимости от местоположения агенства, список возможных департаментов заранее утверждён.

Таблица “Status” предназначена для хранения информации о статусах договоров, а именно “одобрен”, “в обработке”, “отклонён”.

Таблица “Service” предназначена для хранения всех услуг и их описания, а именно название услуги, стоимость, первоначальный срок действия, тип департамента, который может оказывать услугу.

**3.3 Доменные классы**

Доменные классы и их названия полностью совпадают с названиями таблицами, предоставленных на рисунке 3.1, описанных в разделе 3.2. Ниже в таблице 3.1 приведены назначения доменных классов, а в таблице 2.2 приведено название класса, имеющиеся в нём поля, их типы и предназначения. Стоит отметить, что доменные коллекции строятся на базе доменных классов. Вид связи доменных классов – ассоциация. Диаграмма доменных классов приведена на рисунке 3.2

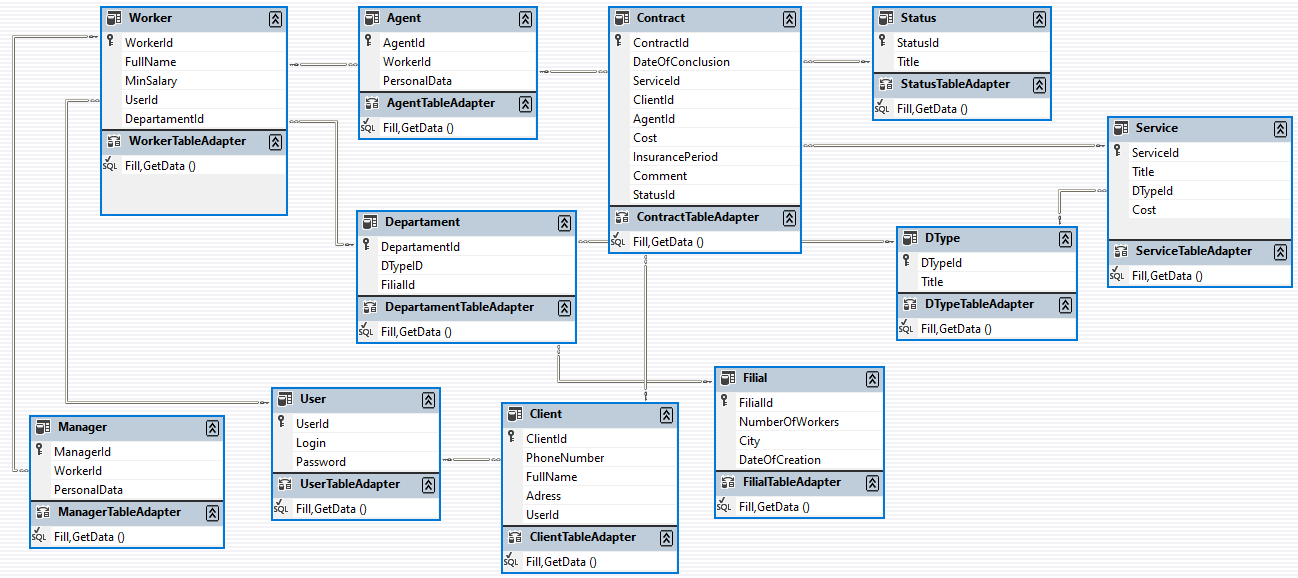


Рисунок 3.2 – Диаграмма доменных классов

Таблица 3.1 – Назначения доменных классов

|  |  |
| --- | --- |
| Исходное название доменного класса | Назначение доменного класса |
| User | Хранение информации о пользователе. |
| Worker | Хранение информации о работнике. |
| Client | Хранение информации о клиенте. |
| Agent | Хранение информации о агенте. |
| Manager | Хранение информации о менеджере. |
| Departament | Хранение информации о департаменте. |
| Filial | Хранение информации о филиале. |
| Contract | Хранение информации о договоре. |
| DType | Хранение информации о типе департамента. |
| Status | Хранение информации о типе статуса договора. |
| Service | Хранение информации о услуге. |

Таблица 3.2 – Описание полей доменных классов

|  |  |  |  |
| --- | --- | --- | --- |
| Название доменного класса | Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
| Agent | int | AgentId | Для хранения уникального номера агента. |
| int | WorkerId | Для хранения номера работника из списка работников. |
| string | PersonalData | Для хранения персональных данных агента. |
| Client | int | ClientId | Для хранения уникального номера клиента. |

Продолжение таблицы 3.2

|  |  |  |  |
| --- | --- | --- | --- |
| Название доменного класса | Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
|  | string | PhoneNumber | Для хранения номера телефона. |
| string | FullName | Для хранения ФИО. |
| string | Adress | Для хранения адреса проживания. |
| int | UserId | Для хранения номера пользователя, к которому привязан клиент. |
| Contract | int | ContractId | Для хранения уникального номера договора |
| string | DateOfConclusion | Для хранения даты регистрации. |
| int | ServiceId | Для хранения номера услуги. |
| int | ClientId | Для хранения номера клиента. |
| int | AgentId | Для хранения номера агента. |
| int | Cost | Для хранения цены оформления. |
| int | InsurancePeriod | Для хранения периода страхования. |
| string | Comment | Для хранения комментария клиента. |
| int | StatusId | Для хранения номера статуса договора. |
| Departament | int | DepartamentId | Для хранения уникального номера департамента. |
| int | DTypeID | Для хранения номера вида департамента. |
| int | FilialId | Для хранения номера филиала. |
| DType | int | DTypeId | Для хранения уникального номера вида департамента. |
| string | Title | Для хранения названия вида департамента. |
| Filial | int | FilialId | Для хранения уникального номера филиала. |

Продолжение таблицы 3.2

|  |  |  |  |
| --- | --- | --- | --- |
| Название доменного класса | Тип данных свойства | Свойство доменного класса | Предназначение свойства доменного класса |
|  | int | NumberOfWorkers | Для хранения количества работников. |
| string | City | Для хранения названия города. |
| string | DateOfCreation | Для хранения даты создания. |
| Manager | int | ManagerId | Для хранения уникального номера менеджера. |
| int | WorkerId | Для хранения номера работника. |
| string | PersonalData | Для хранения персональных данных. |
| Service | int | ServiceId | Для хранения уникального номера услуги. |
| string | Title | Для хранения названия услуги. |
| int | DTypeId | Для хранения номера типа департамента. |
| int | Cost | Для хранения цены услуги. |
| Status | int | StatusId | Для хранения уникального номера статуса договора. |
| string | Title | Для хранения названия статуса договора. |
| User | int | UserId | Для хранения уникального номера пользователя. |
| string | Login | Для хранения логина. |
| string | Password | Для хранения пароля. |
| Worker | int | WorkerId | Для хранения уникального номера работника. |
|  | string | FullName | Для хранения ФИО. |
|  | int | MinSalary | Для хранения минимальной зп. |
| int | UserId | Для хранения номера пользователя. |
| int | DepartamentId | Для хранения номера департамента. |

Ниже в таблицах 3.3 и 3.4 приведено описание перечислений, используемых доменными классами.

Таблица 3.3 – Описание назначения перечислений

|  |  |  |
| --- | --- | --- |
| Название перечисления | Тип | Описание |
| Role | int | Хранит роль пользователя. |

Таблица 3.4 – Описание констант перечислений

|  |  |  |  |
| --- | --- | --- | --- |
| Название перечисления | Название константы | Значение константы | Описание константы |
| Role | Client | 1 | Право доступа к системе – клиент. |
| Agent | 2 | Право доступа к системе – агент. |
| Manager | 3 | Право доступа к системе – менеджер. |

**3.4 Описание интерфейса**

Графический интерфейс — система средств для взаимодействия пользователя с компьютером, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т. п.).

В отличие от интерфейса командной строки, в GUI пользователь имеет произвольный доступ (с помощью устройств ввода — клавиатуры, мыши, джойстика и т. п.) ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. Чаще всего элементы интерфейса в GUI реализованы на основе метафор и отображают их назначение и свойства, что облегчает понимание и освоение программ неподготовленными пользователями.

Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации.

Для входа в систему была реализована форма авторизации пользователя рисунок 3.3. Для входа в систему требуется логин и пароль, хеш которого хранится в базе данных в таблице “User”.

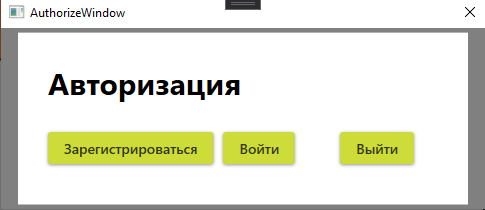


Рисунок 3.3 – Окно авторизации

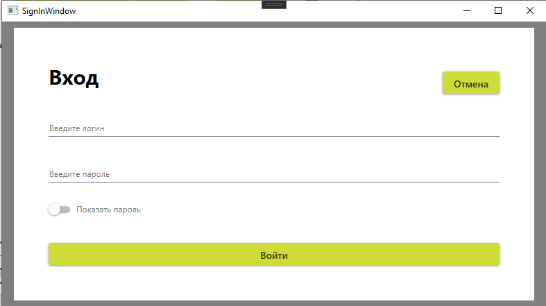


Рисунок 3.4 – Окно входа

Если введенная пара логин-пароль верна, то мы получаем данные о пользователе и его правах доступа, которые хранятся в таблице “User”.

Опишем каждое окно роли программы.

1. Окно клиента

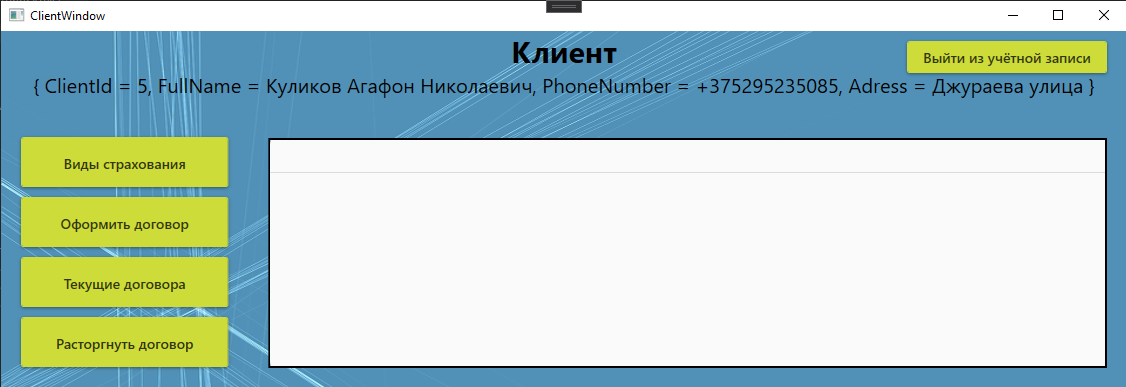


Рисунок 3.5 – Окно клиента, вкладка “Главная”

Во вкладке окна “Главная” окна “Клиент”, показанной на рисунке 3.5, пользователь может ознакомиться с перечнем услуг, оказываемых страховым агенством, найти необходимую ему услугу с помощью удобного фильтра, располагающегося на правой панели. Также он может выбрать вкладку “оформить договор”, где можно выбрать филиал, департамент, страхового агента, услугу, период страхования, а также указать комментарий и ознакомиться с результирующей суммой. Каталог услуг формируется из таблиц, для получения информации из которых используются доменные классы “Service”, “Departament” и “DType”. После того, как клиент определится с выбранными услугами он может ознакомиться с итоговой стоимостью заказа, указать примечание к заказу и нажать на кнопку “Оформить заказ”. После оформления заказ сразу добавляется в таблицу “Contract”.

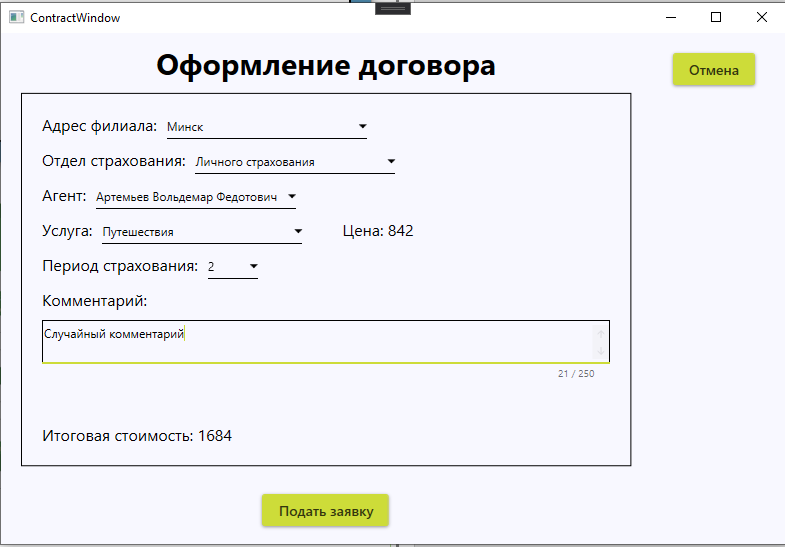


Рисунок 3.6 – Окно клиента, вкладка “Оформить договор”

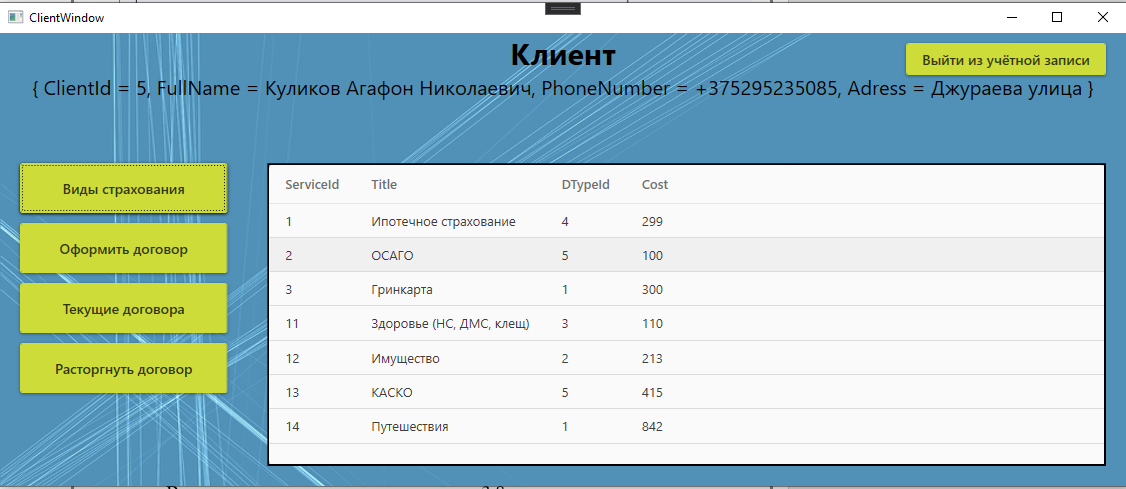


Рисунок 3.7 – Окно клиента, вкладка “Виды страхования”

В окне, изображённом на рисунке 3.7, клиент сможет увидеть список доступных услуг.

1. Окно агента

В окне, показанном на рисунке 3.8, агент может увидеть поступившие на него заказы и обработать их. Заказы хранятся в таблице “Contract”, а услуги заказа в таблице “Service”. Для согласования заказа агенту доступен телефон заказчика, его ФИО и список заказанных услуг. Сведения о заказчике хранятся в таблице “Client”. При подтверждении или отмене заказа программа изменяет в записи таблицы “Contract” поле “StatusId”.

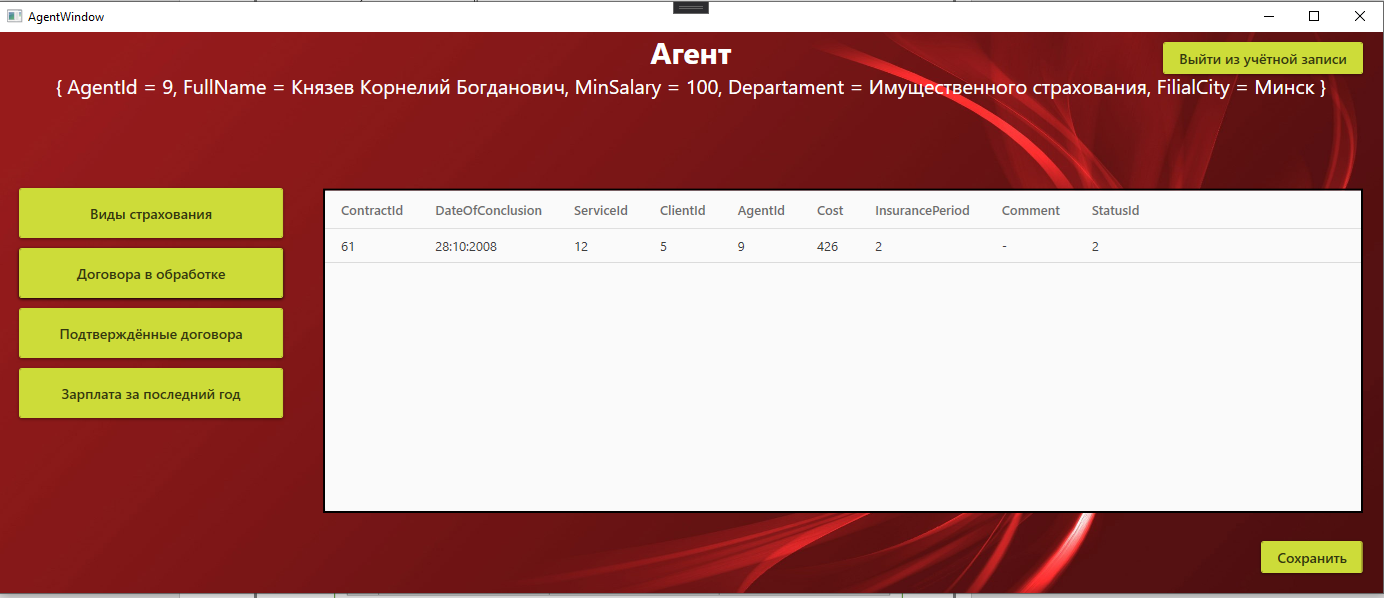


Рисунок 3.8 – Окно агента

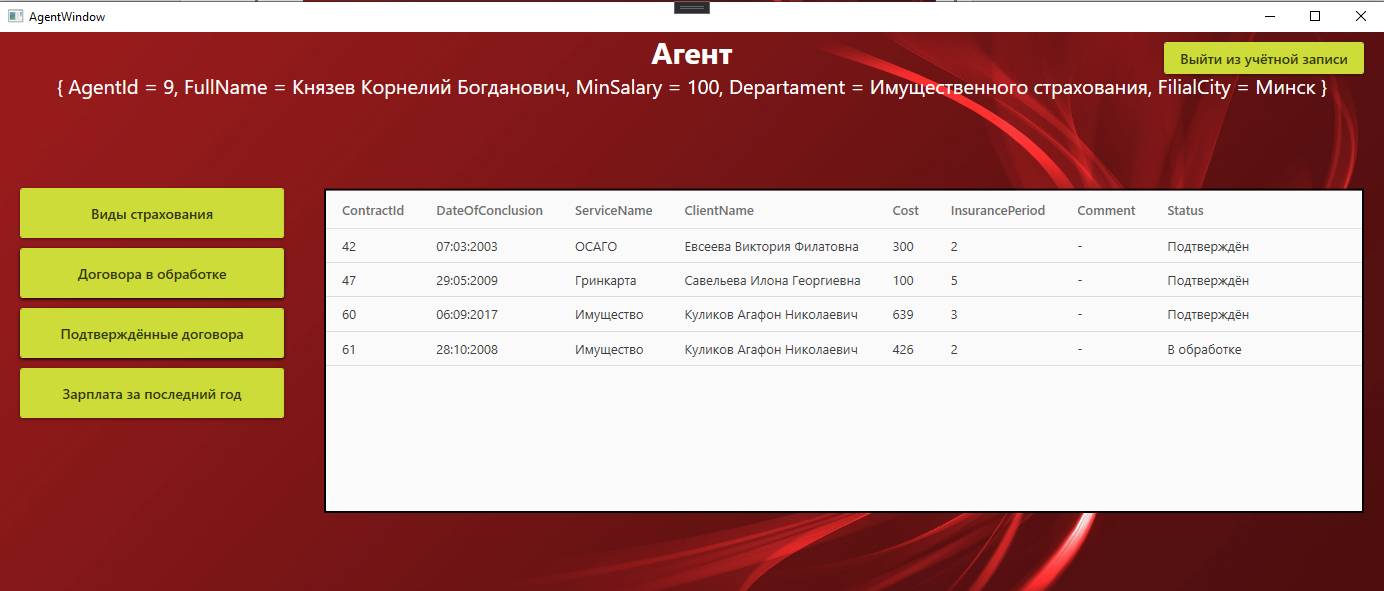


Рисунок 3.9 – Окно с подтверждёнными договорами

1. Окно менеджера

После авторизации в программе менеджер увидит перед собой окно, показанное на рисунке 3.10. Ему доступны операции просмотра отчётности (рисунки 3.11, 3.12), просмотра списка сотрудников, вынесения премирования или депремирования. Информация для этой роли собирается из всех таблиц базы данных

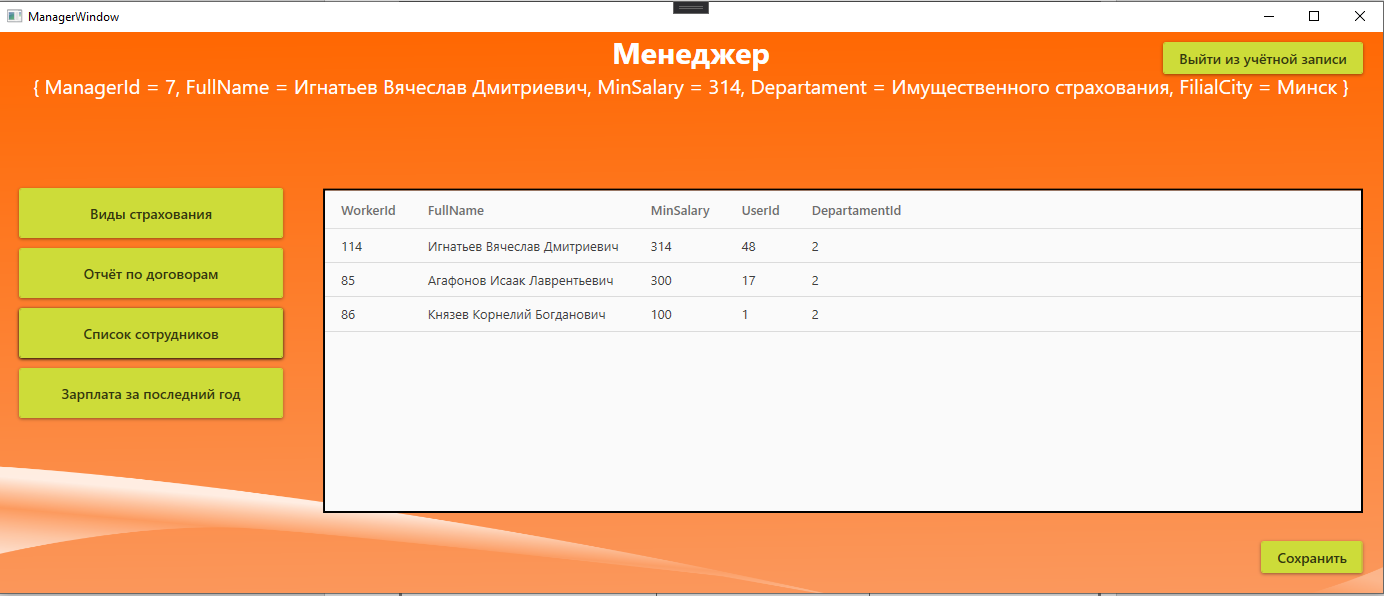


Рисунок 3.10 – Окно менеджера

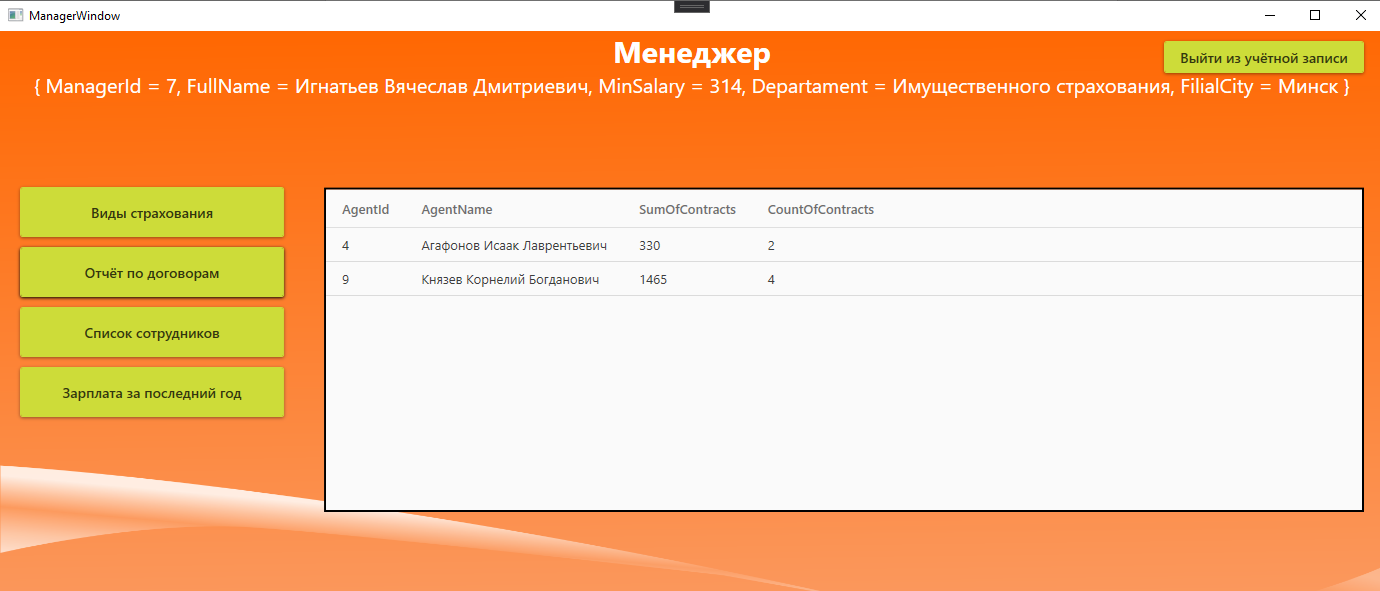


Рисунок 3.11 – Окно с отчётом по договорам

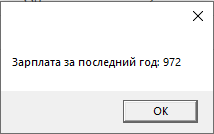


Рисунок 3.12 – Окно со статистикой за последний год

Теперь опишем окно регистрации, которое открывается нажатием на кнопку “Зарегистрироваться” формы, показанной выше на рисунке 3.3. Изображение этой формы показано на рисунке 3.13. В этом окне можно увидеть следующие поля для заполнения: логин, пароль, поле повторного ввода пароля, поле ввода ФИО, телефонного номера и адреса проживания. После заполнения всех вышеперечисленных полей и нажатия на кнопку зарегистрироваться, в случае если их валидация пройдёт успешно, то регистрация будет завершена и в таблицы “User” и “Client” будут добавлены записи со сведениями о новом пользователе. Информацию о таблицах “User” и “Client” можно найти на странице 15 пункта 3.2.

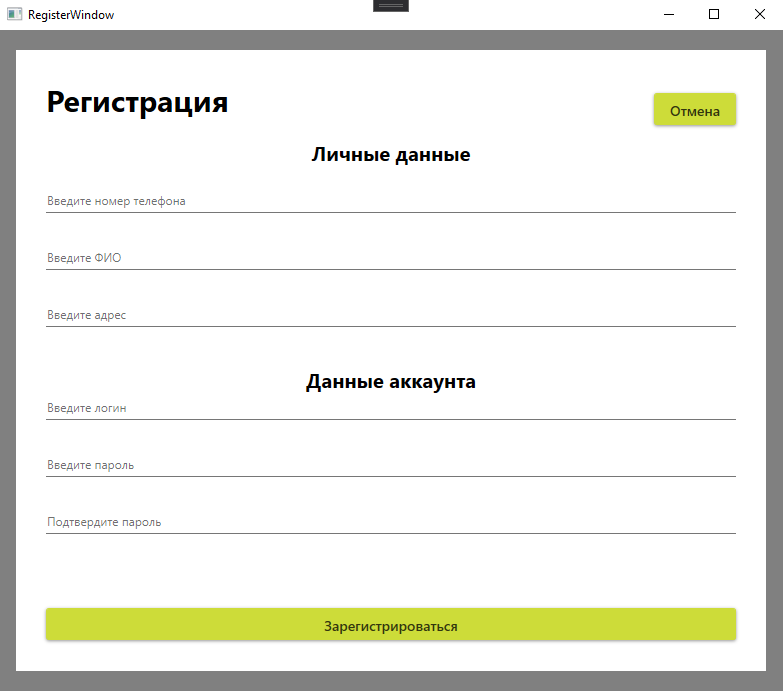


Рисунок 3.13 – Окно регистрации

**4 ТЕСТИРОВАНИЕ**

**4.1 Верификация**

На основе требований к проектируемому программному обеспечению, приведёнными в пункте 1.3, сформирована таблица 4.1, в которой показано какие формы отвечают за реализацию этих требований.

Таблица 4.1 – Верификация

|  |  |
| --- | --- |
| Требование к проектируемому ПО | Окна, реализующие требования |
| Ведение справочников | AgentWindow |
| Заказ услуг | ClientWindow |
| Обработка заказов | AgentWindow |
| Выполнение заказа | AgentWindow |
| Оплата заказа |
| Закрытие заказа |
| Создание отчётов | ManagerWindow |
| Статистика |

Таким образом, исходя из таблицы 4.1 видно, что все основные задачи курсового проектирования выполнены.

**4.2 Обработка исключительных ситуаций**

Протестируем вариант запуска программы без базы данных. В результате соблюдения указанных условий программа открывает при запуске окно, показанное на рисунке 4.1.

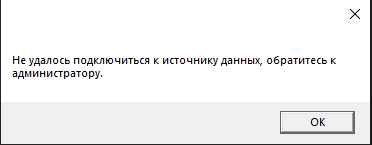


Рисунок 4.1 – Окно ошибки подключения к БД

Если удалить все файлы ресурсов приложения и затем его запустить, то перед запуском формы авторизации (рисунок 3.3) будет показано предупреждающее сообщение (рисунок 4.2), но окно авторизации всё же откроется.

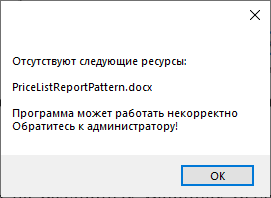


Рисунок 4.2 – Ошибка наличия ресурсов программы

Протестируем тот случай, когда пользователь формирует отчёт без необходимого ресурсного файла. Результат тестирования показан на рисунке 4.3.

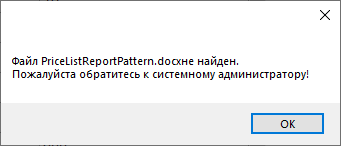


Рисунок 4.3 – Ошибка наличия ресурсного файла программы

Протестируем окно регистрации (рисунок 4.4). Сначала попробуем зарегистрироваться, не заполнив ни одно из полей. В результате увидим окно, показанное на рисунке 4.5. Также тестирование завершалось ошибкой при несовпадении текста поля “пароль” и “пароль (ещё раз)”, отсутствии содержимого полей “ФИО” и “Номер телефона”.

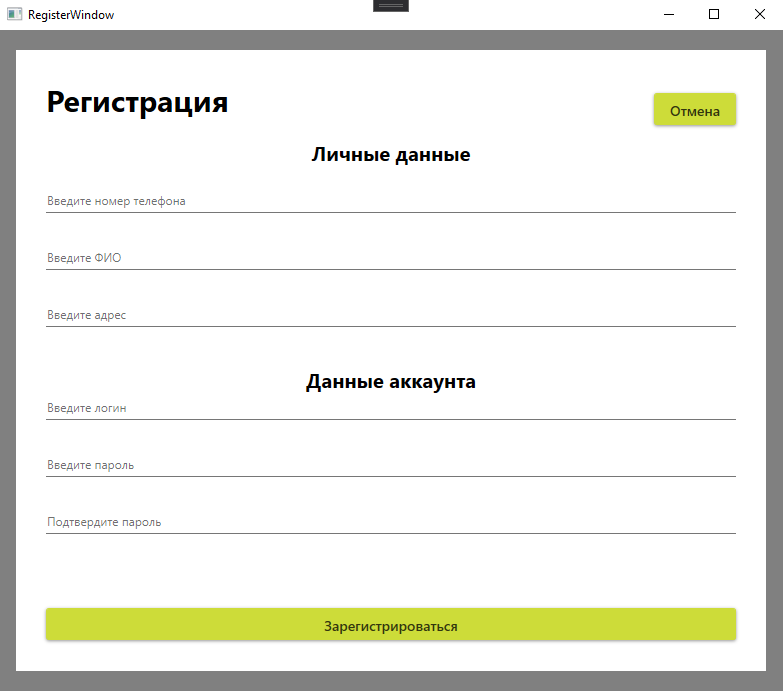


Рисунок 4.4 – Окно регистрации

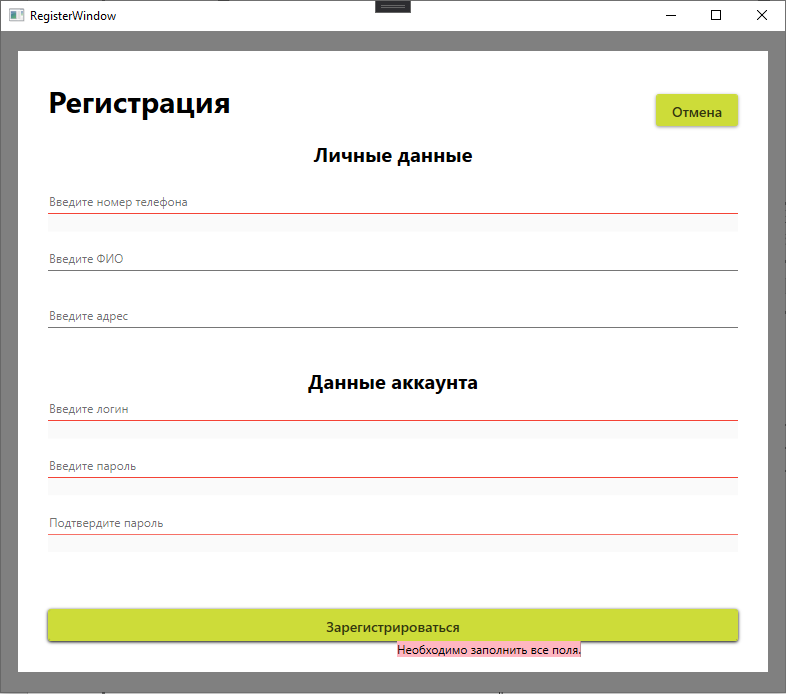


Рисунок 4.5 – Ошибка регистрации

Попробуем удалить связные данные в справочнике. При удалении категории услуги, которая ещё используется в услугах, получим ошибку, показанную на рисунке 4.6.

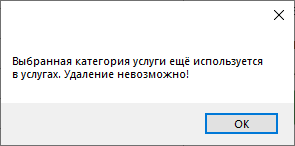


Рисунок 4.6 – Ошибка удаления категории услуги

При попытке удаления услуги, которая используется в прейскуранте, получим ошибку, показанную на рисунке 4.7.

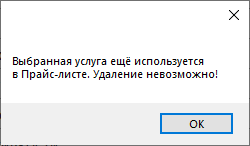


Рисунок 4.7 – Ошибка удаления услуги

В ходе запуска на машинах с различными конфигурациями оборудования были установлены минимальные системные требования:

* операционная система: Windows XP или выше;
* процессор Intel Pentium IV с тактовой частотой 1.2 ГГц и выше;
* ОЗУ: 256 МБ и более;
* свободное место на жёстком диске: 15 МБ и более.

**ЗАКЛЮЧЕНИЕ**

В результате курсового проектирования был создан продукт, цель которого – автоматизация работы страхового агенства. Созданная программа обладает функциями, необходимыми для формирования клиентом заказа, его продвижения и организации работы сотрудников, сбора статистики за определённый период, создания отчётов и их последующего экспортирования в новый документ текстового редактора MS Word. В процессе создания продукта были пройдены такие этапы разработки, как постановка задачи, составление логической модели предметной области, создание и реализация алгоритма работы программы с последующим тестированием.

Продукт был разработан, отлажен и протестирован. Поставленные в курсовой работе задачи были полностью выполнены, однако всё ещё есть некоторые возможности, реализуя которые можно добиться более углубленного моделирования предметной области.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Рихтер, Джеффри CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C# / Джеффри Рихтер. - М.: Питер, 2013. - 928 c
2. Джозеф Альбахари и Бен Альбахари. C# 7.0. Карманный справочник. – «O’Reilly Media, Inc», 2017. – 156 с.
3. Эндрю Троелсен. Язык программирования C# 5.0 и платформа .NET 4.5, 6-е издание. — М.: «Вильямс», 2013. — 792 с.
4. Microsoft.Office.Interop.Word Namespace [Электронный ресурс]. – URL: https://docs.microsoft.com/ru-ru/dotnet/api/microsoft.office.interop.word?view=word-pia, свободный. – Загл. с экрана.
5. Отображение значения Enum в C# на русском [Электронный ресурс]. – URL: https://shwanoff.ru/enum-to-string/, свободный. – Загл. с экрана.

**ПРИЛОЖЕНИЕ А**

using CourseApp.Classes;

using CourseApp.Services;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Controls.Primitives;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace CourseApp.Accounts.AgentWindow

{

/// <summary>

/// Логика взаимодействия для AgentWin.xaml

/// </summary>

public partial class AgentWindow : Window

{

protected object cellValue;

protected Agent agent;

public AgentWindow(Agent agent)

{

InitializeComponent();

this.agent = agent;

AgentInfoString.Text = AgentService.GetFullInfoById(agent.AgentId);

}

// 1

private void ShowTypesOfInsurance(object sender, RoutedEventArgs e)

{

HideSaveButton();

DataGrid.ItemsSource = ServiceService.GetServicesByAgentId(agent.AgentId);

}

// 2

private void ContractsInProcessing(object sender, RoutedEventArgs e)

{

ShowSaveButton();

DataGrid.ItemsSource = ContractService.GetContractsInProcessingByAgentId(agent.AgentId);

for (int i = 0; i < DataGrid.Columns.Count - 1; ++i)

DataGrid.Columns[i].IsReadOnly = true;

}

// 3

private void ConfirmedContracts(object sender, RoutedEventArgs e)

{

HideSaveButton();

DataGrid.ItemsSource = ContractService.GetFullInfoByAgentId(agent.AgentId);

}

// 4

private void SalaryForTheLastYear(object sender, RoutedEventArgs e)

{

HideSaveButton();

string salaryStr = "Зарплата за последний год: " +

AgentService.GetSalary(agent.AgentId);

MessageBox.Show(salaryStr);

}

// buttons

private void Save(object sender, RoutedEventArgs e)

{

if (DataGrid.ItemsSource is IEnumerable<Contract> contracts)

foreach (Contract contract in contracts)

ContractService.Update(contract);

MessageBox.Show("Сохранение выполнено успешно.");

}

private void Exit(object sender, RoutedEventArgs e)

{

var authorize = new AuthorizeWindow();

authorize.Show();

this.Close();

}

// menu

private void ShowSaveButton()

{

DataGrid.IsReadOnly = false;

SaveButton.Visibility = Visibility.Visible;

}

private void HideSaveButton()

{

DataGrid.IsReadOnly = true;

SaveButton.Visibility = Visibility.Hidden;

}

}

}

using CourseApp.Classes;

using CourseApp.Services;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace CourseApp.Accounts.ClientWindow.ContractWindow

{

/// <summary>

/// Логика взаимодействия для ContractWindow.xaml

/// </summary>

public partial class ContractWindow : Window

{

protected Client client;

protected Filial filial;

protected Departament departament;

protected DType dType;

protected Agent agent;

protected Service service;

protected int insuransePeriod;

protected int cost;

protected string comment;

public ContractWindow(Client client)

{

InitializeComponent();

this.client = client;

filial = new Filial();

departament = new Departament();

dType = new DType();

agent = new Agent();

service = new Service();

comment = "-";

}

private void Exit(object sender, RoutedEventArgs e) => this.Close();

// filial

private void EnterFilial(object sender, EventArgs e)

{

FilialComboBox.ItemsSource = FilialService.GetCities();

}

private void SelectFilial(object sender, EventArgs e)

{

if (FilialComboBox.SelectedItem != null)

{

filial = FilialService.GetFilialByCity((string) FilialComboBox.SelectedItem);

ClearDepartament();

ClearAgent();

ClearService();

}

}

// departament

private void EnterDepartament(object sender, EventArgs e)

{

DepartamentComboBox.ItemsSource = DepartamentService.GetDepartamentTitlesByFilial(filial);

}

private void SelectDepartament(object sender, EventArgs e)

{

if (DepartamentComboBox.SelectedItem != null)

{

departament = DepartamentService.GetDepartamentByTitleAndFilialId

(

(string) DepartamentComboBox.SelectedItem,

filial.FilialId

);

dType = DTypeService.GetDtypeById(departament.DTypeID);

ClearAgent();

ClearService();

UpdateCost();

}

}

// agent

private void EnterAgent(object sender, EventArgs e)

{

AgentComboBox.ItemsSource = AgentService.GetFullNamesByDepartamentId(departament.DepartamentId);

}

private void SelectAgent(object sender, EventArgs e)

{

if (AgentComboBox.SelectedItem != null)

{

agent = AgentService.GetAgentByFullnameAndDepartamentId

(

(string)AgentComboBox.SelectedItem,

departament.DepartamentId

);

}

}

// service

private void EnterService(object sender, EventArgs e)

{

ServiceComboBox.ItemsSource = ServiceService.GetNamesByDtypeId(dType.DTypeId);

}

private void SelectService(object sender, EventArgs e)

{

if (ServiceComboBox.SelectedItem != null)

{

service = ServiceService.GetServiceByTitleAndDTypeId

(

(string) ServiceComboBox.SelectedItem,

dType.DTypeId

);

UpdateCost();

}

}

// period

private void SelectPeriod(object sender, EventArgs e)

{

if (PeriodComboBox.SelectedItem != null)

{

insuransePeriod = int.Parse(((TextBlock)PeriodComboBox.SelectedItem).Text);

UpdateCost();

}

}

// comment

private void CommentChanged(object sender, EventArgs e)

{

comment = CommentTextBox.Text;

}

// registrate button

private void RegistrateContract(object sender, RoutedEventArgs e)

{

if (service.ServiceId == 0 ||

client.ClientId == 0 ||

agent.AgentId == 0 ||

cost == 0 ||

insuransePeriod == 0)

{

MessageBox.Show("Заполните все поля.");

}

else

{

ContractService.Add

(

new Contract

(

GetDate(),

service.ServiceId,

client.ClientId,

agent.AgentId,

cost,

insuransePeriod,

comment,

2

)

);

MessageBox.Show("Договор отправлен на рассмотрение, ожидайте подтверждения.");

this.Close();

}

}

private void ClearDepartament()

{

DepartamentComboBox.Text = "";

DepartamentComboBox.ItemsSource = null;

departament = new Departament();

}

private void ClearAgent()

{

AgentComboBox.Text = "";

AgentComboBox.ItemsSource = null;

agent = new Agent();

}

private void ClearService()

{

ServiceComboBox.Text = "";

ServiceComboBox.ItemsSource = null;

service = new Service();

UpdateCost();

}

private void UpdateCost()

{

cost = insuransePeriod \* service.Cost;

ServiceCostTextBlock.Text = service.Cost.ToString();

ContractCost.Text = cost.ToString();

}

private string GetDate()

{

Random random = new Random();

string day = (random.Next() % 31 + 1).ToString();

string month = (random.Next() % 12 + 1).ToString();

string year = (random.Next() % 22 + 2000).ToString();

if (day.Length < 2)

day = "0" + day;

if (month.Length < 2)

month = "0" + month;

while (year.Length < 4)

year = "0" + year;

return day + ":" + month + ":" + year;

}

}

}

using CourseApp.Classes;

using CourseApp.Services;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace CourseApp.Accounts.ClientWindow.TerminateContractWindow

{

/// <summary>

/// Логика взаимодействия для TerminateContractWindow.xaml

/// </summary>

public partial class TerminateContractWindow : Window

{

protected Client client;

protected Contract contract;

public TerminateContractWindow(Client client)

{

InitializeComponent();

this.client = client;

TerminateContractsDataGrid.ItemsSource = ContractService.GetFullInfoByClientId(client.ClientId);

}

private void EnterContract(object sender, EventArgs e)

{

ContractIdComboBox.ItemsSource = ContractService.GetContractIdByClientId(client.ClientId);

}

private void SelectContract(object sender, EventArgs e)

{

if (ContractIdComboBox.SelectedItem != null)

{

int contractId = (int) ContractIdComboBox.SelectedItem;

contract = ContractService.GetContractById(contractId);

}

}

private void TerminateContract(object sender, RoutedEventArgs e)

{

if (contract != null)

{

MessageBox.Show("Договор расторгнут.");

ContractService.Remove(contract);

this.Close();

}

else

{

MessageBox.Show("Выберите номер договора.");

}

}

private void Back(object sender, RoutedEventArgs e) => this.Close();

}

}

using CourseApp.Classes;

using CourseApp.Services;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using CourseApp.Accounts.ClientWindow.TerminateContractWindow;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace CourseApp.Accounts.ClientWindow

{

/// <summary>

/// Логика взаимодействия для ClientWin.xaml

/// </summary>

public partial class ClientWindow : Window

{

protected Client client;

public ClientWindow(Client client)

{

InitializeComponent();

this.client = client;

ClientInfoString.Text = ClientService.GetFullInfoById(client.ClientId);

}

private void Exit(object sender, RoutedEventArgs e)

{

var authorize = new AuthorizeWindow();

authorize.Show();

this.Close();

}

private void ShowTypesOfInsurance(object sender, RoutedEventArgs e)

{

DataGrid.ItemsSource = ServiceService.GetServices();

}

private void ConcludeContract(object sender, RoutedEventArgs e)

{

var contractWindow = new ContractWindow.ContractWindow(client);

contractWindow.ShowDialog();

}

private void CurrentContracts(object sender, RoutedEventArgs e)

{

DataGrid.ItemsSource = ContractService.GetFullInfoByClientId(client.ClientId);

}

private void TerminateContract(object sender, RoutedEventArgs e)

{

if (ContractService.GetContractIdByClientId(client.ClientId).Any())

{

var terminateContractWindow = new TerminateContractWindow.TerminateContractWindow(client);

terminateContractWindow.ShowDialog();

}

else

{

MessageBox.Show("На текущего клиента не оформлено ни одного договора.");

}

}

}

}

using CourseApp.Classes;

using CourseApp.Services;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace CourseApp.Accounts.ManagerWindow

{

/// <summary>

/// Логика взаимодействия для ManagerWin.xaml

/// </summary>

public partial class ManagerWindow : Window

{

protected Manager manager;

public ManagerWindow(Manager manager)

{

InitializeComponent();

this.manager = manager;

ManagerInfoString.Text = ManagerService.GetFullInfoById(manager.ManagerId);

}

private void ShowTypesOfInsurance(object sender, RoutedEventArgs e)

{

ShowSaveButton();

DataGrid.ItemsSource = ServiceService.GetServices();

for (int i = 0; i < DataGrid.Columns.Count - 1; ++i)

DataGrid.Columns[i].IsReadOnly = true;

}

private void ReportOnContracts(object sender, RoutedEventArgs e)

{

HideSaveButton();

DataGrid.ItemsSource = ContractService.GetReportByManagerId(manager.ManagerId);

}

private void ListOfWorkers(object sender, RoutedEventArgs e)

{

ShowSaveButton();

DataGrid.ItemsSource = WorkerService.GetWorkersByDepartamentId(

ManagerService.GetDepartamentIdByManagerId(manager.ManagerId));

DataGrid.Columns[0].IsReadOnly = true;

DataGrid.Columns[1].IsReadOnly = true;

DataGrid.Columns[3].IsReadOnly = true;

DataGrid.Columns[4].IsReadOnly = true;

}

private void SalaryForTheLastYear(object sender, RoutedEventArgs e)

{

HideSaveButton();

MessageBox.Show("Зарплата за последний год: " +

ManagerService.GetSalary(manager.ManagerId));

}

private void Save(object sender, RoutedEventArgs e)

{

if (DataGrid.ItemsSource is IEnumerable<Service> services)

foreach (Service service in services)

ServiceService.Update(service);

if (DataGrid.ItemsSource is IEnumerable<Worker> workers)

foreach (Worker worker in workers)

WorkerService.Update(worker);

MessageBox.Show("Сохранение данных выполнено успешно.");

}

private void Exit(object sender, RoutedEventArgs e)

{

var authorize = new AuthorizeWindow();

authorize.Show();

this.Close();

}

// menu

private void ShowSaveButton()

{

DataGrid.IsReadOnly = false;

SaveButton.Visibility = Visibility.Visible;

}

private void HideSaveButton()

{

DataGrid.IsReadOnly = true;

SaveButton.Visibility = Visibility.Hidden;

}

}

}

using CourseApp.Services;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data.OleDb;

using System.Globalization;

using System.Linq;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading.Tasks;

using System.Windows.Controls;

namespace CourseApp.Classes

{

class RegisterInfoOfClient : IDataErrorInfo

{

public string Number { get; set; }

public string ClientName { get; set; }

public string Adress { get; set; }

public string Login { get; set; }

public string Password1 { get; set; }

public string Password2 { get; set; }

public RegisterInfoOfClient()

{

Number = string.Empty;

ClientName = string.Empty;

Adress = string.Empty;

Login = string.Empty;

Password1 = string.Empty;

Password2 = string.Empty;

}

public bool IsValid()

{

bool result = true;

Type type = typeof(RegisterInfoOfClient);

var properties = type.GetProperties();

foreach (var property in properties)

result &= (Validate(property.Name) == string.Empty);

return result;

}

public string this[string columnName] => Validate(columnName);

public string Validate(string name)

{

string error = string.Empty;

switch (name)

{

case "Number":

{

string pattern = @"\+[0-9]{12}";

if (Number == string.Empty)

error = " ";

else if (!Regex.IsMatch(Number, pattern))

error = "Некорректный номер";

break;

}

case "ClientName":

{

break;

}

case "Adress":

{

break;

}

case "Login":

{

if (Login == string.Empty)

error = " ";

else if (UserService.ContainsLogin(Login))

error = "Логин уже занят";

break;

}

case "Password1":

{

if (Password1 == string.Empty)

error = " ";

else if (Password1 != Password2 && Password2 != string.Empty)

error = "Пароли не совпадают";

break;

}

case "Password2":

{

if (Password2 == string.Empty)

error = " ";

else if (Password1 != Password2)

error = "Пароли не совпадают";

break;

}

}

return error;

}

public string Error => throw new NotImplementedException();

}

}

using CourseApp.Accounts.ClientWindow;

using CourseApp.Classes;

using CourseApp.Services;

using System;

using System.Collections.Generic;

using System.Data.OleDb;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace CourseApp.AuthorizeWindows.Register

{

/// <summary>

/// Логика взаимодействия для RegisterWindow.xaml

/// </summary>

public partial class RegisterWindow : Window

{

public int CodeOfUser { get; set; }

private RegisterInfoOfClient registerInfo;

public RegisterWindow()

{

InitializeComponent();

registerInfo = new RegisterInfoOfClient();

this.DataContext = registerInfo;

}

private void Back(object sender, RoutedEventArgs e)

{

var authorize = new AuthorizeWindow();

authorize.Show();

this.Close();

}

private void Register(object sender, RoutedEventArgs e)

{

if (registerInfo.IsValid())

{

RegPopUp.Visibility = Visibility.Hidden;

User user = new User(registerInfo.Login, registerInfo.Password1);

UserService.Add(user);

CodeOfUser = UserService.GetId(user);

Client client = new Client(registerInfo.Number, registerInfo.ClientName, registerInfo.Adress, CodeOfUser);

ClientService.Add(client);

var clientWindow = new ClientWindow(ClientService.GetClientByUserId(user.UserId));

clientWindow.Show();

this.Close();

}

else

{

RegPopUpText.Visibility = Visibility.Visible;

RegPopUp.IsOpen = true;

}

}

private void PopUpLeave(object sender, MouseEventArgs e)

{

if (RegPopUpText.Visibility == Visibility.Visible)

RegPopUp.IsOpen = false;

}

}

}

using CourseApp.Accounts.AgentWindow;

using CourseApp.Accounts.ClientWindow;

using CourseApp.Accounts.ManagerWindow;

using CourseApp.Classes;

using CourseApp.Services;

using System;

using System.Collections.Generic;

using System.Data.OleDb;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace CourseApp.AuthorizeWindows.SignIn

{

/// <summary>

/// Логика взаимодействия для SignInWindow.xaml

/// </summary>

public partial class SignInWindow : Window

{

public SignInWindow()

{

InitializeComponent();

}

public void SignIn(User user)

{

user.UserId = UserService.GetId(user);

Role role = UserService.GetRole(user.UserId);

switch (role)

{

case Role.Client:

{

var clientWindow = new ClientWindow(ClientService.GetClientByUserId(user.UserId));

clientWindow.Show();

break;

}

case Role.Agent:

{

var agentWindow = new AgentWindow(AgentService.GetAgentByUserId(user.UserId));

agentWindow.Show();

break;

}

case Role.Manager:

{

var managerWindow = new ManagerWindow(ManagerService.GetManagerByUserId(user.UserId));

managerWindow.Show();

break;

}

}

}

private void CangeVisibleOfPassword(object sender, RoutedEventArgs e)

{

if (TToggle.IsChecked == true)

{

TPassword.Visibility = Visibility.Hidden;

TTextPassword.Visibility = Visibility.Visible;

TTextPassword.Text = TPassword.Password;

}

else

{

TTextPassword.Visibility = Visibility.Hidden;

TPassword.Visibility = Visibility.Visible;

TPassword.Password = TTextPassword.Text;

}

}

private void SignIn(object sender, RoutedEventArgs e)

{

string login = TLogin.Text;

string password;

if (TToggle.IsChecked == true)

password = TTextPassword.Text;

else

password = TPassword.Password;

User user = new User(login, password);

if (UserService.Contains(user))

{

SignIn(user);

this.Close();

}

else

{

TLogin.Foreground = new SolidColorBrush(Color.FromRgb(255, 0, 0));

TPassword.Foreground = new SolidColorBrush(Color.FromRgb(255, 0, 0));

TTextPassword.Foreground = new SolidColorBrush(Color.FromRgb(255, 0, 0));

}

}

private void Back(object sender, RoutedEventArgs e)

{

var authorize = new AuthorizeWindow();

authorize.Show();

this.Close();

}

}

}

using CourseApp.AuthorizeWindows.Register;

using CourseApp.AuthorizeWindows.SignIn;

using CourseApp.Classes;

using CourseApp.Repository;

using CourseApp.Services;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

namespace CourseApp

{

/// <summary>

/// Логика взаимодействия для AuthorizeWindow.xaml

/// </summary>

public partial class AuthorizeWindow : Window

{

public AuthorizeWindow()

{

InitializeComponent();

}

private void Register(object sender, RoutedEventArgs e)

{

var register = new RegisterWindow();

register.Show();

this.Close();

}

private void SignIn(object sender, RoutedEventArgs e)

{

var signIn = new SignInWindow();

signIn.Show();

this.Close();

}

private void Exit(object sender, RoutedEventArgs e)

{

this.Close();

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class Agent

{

public int AgentId { get; set; }

public int WorkerId { get; set; }

public string PersonalData { get; set; }

public Agent() { }

public Agent(int workerId, string personalData)

{

WorkerId = workerId;

PersonalData = personalData;

}

public override string ToString()

{

return AgentId + " " +

WorkerId + " " +

PersonalData;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class Client

{

public int ClientId { get; set; }

public string PhoneNumber { get; set; }

public string FullName { get; set; }

public string Adress { get; set; }

public int UserId { get; set; }

public Client() { }

public Client(string phoneNumber, string fullName, string adress, int userId)

{

PhoneNumber = phoneNumber;

FullName = fullName;

Adress = adress;

UserId = userId;

}

public override string ToString()

{

return ClientId + " " +

PhoneNumber + " " +

FullName + " " +

Adress + " " +

UserId;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class Contract

{

public int ContractId { get; set; }

public string DateOfConclusion { get; set; }

public int ServiceId { get; set; }

public int ClientId { get; set; }

public int AgentId { get; set; }

public int Cost { get; set; }

public int InsurancePeriod { get; set; }

public string Comment { get; set; }

public int StatusId { get; set; }

public Contract() { }

public Contract(string dateOfConclusion, int serviceId, int clientId, int agentId, int cost, int insurancePeriod, string comment, int statusId)

{

DateOfConclusion = dateOfConclusion;

ServiceId = serviceId;

ClientId = clientId;

AgentId = agentId;

Cost = cost;

InsurancePeriod = insurancePeriod;

Comment = comment;

StatusId = statusId;

}

public override string ToString()

{

return ContractId + " " +

DateOfConclusion + " " +

ServiceId + " " +

ClientId + " " +

AgentId + " " +

Cost + " " +

InsurancePeriod + " " +

Comment + " " +

StatusId;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class Departament

{

public int DepartamentId { get; set; }

public int DTypeID { get; set; }

public int FilialId { get; set; }

public Departament() { }

public Departament(int dTypeID, int filialId)

{

DTypeID = dTypeID;

FilialId = filialId;

}

public override string ToString()

{

return DepartamentId + " " +

DTypeID + " " +

FilialId;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class DType

{

public int DTypeId { get; set; }

public string Title { get; set; }

public DType() { }

public DType(string title)

{

Title = title;

}

public override string ToString()

{

return DTypeId + " " +

Title;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class Filial

{

public int FilialId { get; set; }

public int NumberOfWorkers { get; set; }

public string City { get; set; }

public string DateOfCreation { get; set; }

public Filial() { }

public Filial(int numberOfWorkers, string city, string dateOfCreation)

{

NumberOfWorkers = numberOfWorkers;

City = city;

DateOfCreation = dateOfCreation;

}

public override string ToString()

{

return FilialId + " " +

NumberOfWorkers + " " +

City + " " +

DateOfCreation;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class Manager

{

public int ManagerId { get; set; }

public int WorkerId { get; set; }

public string PersonalData { get; set; }

public Manager() { }

public Manager(int workerId, string personalData)

{

WorkerId = workerId;

PersonalData = personalData;

}

public override string ToString()

{

return ManagerId + " " +

WorkerId + " " +

PersonalData;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

enum Role

{

Client,

Agent,

Manager

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class Service

{

public int ServiceId { get; set; }

public string Title { get; set; }

public int DTypeId { get; set; }

public int Cost { get; set; }

public Service() { }

public Service(string title, int dTypeId, int cost)

{

Title = title;

DTypeId = dTypeId;

Cost = cost;

}

public override string ToString()

{

return ServiceId + " " +

Title + " " +

DTypeId + " " +

Cost;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class Status

{

public int StatusId { get; set; }

public string Title { get; set; }

public Status() { }

public Status(string title)

{

Title = title;

}

public override string ToString()

{

return StatusId + " " +

Title;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class User

{

public int UserId { get; set; }

public string Login { get; set; }

public string Password { get; set; }

public User() { }

public User(string login, string password)

{

Login = login;

Password = password;

}

public override string ToString()

{

return UserId + " " +

Login + " " +

Password;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Classes

{

public class Worker

{

public int WorkerId { get; set; }

public string FullName { get; set; }

public int MinSalary { get; set; }

public int UserId { get; set; }

public int DepartamentId { get; set; }

public Worker() { }

public Worker(string fullName, int minSalary, int userId, int departamentId)

{

FullName = fullName;

MinSalary = minSalary;

UserId = userId;

DepartamentId = departamentId;

}

public override string ToString()

{

return WorkerId + " " +

FullName + " " +

MinSalary + " " +

UserId + " " +

DepartamentId;

}

}

}

using System;

using System.Collections.Generic;

using System.Data.OleDb;

using System.Configuration;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace CourseApp

{

static class DB

{

static private readonly string connectStr = ConfigurationManager.ConnectionStrings["DefaultConnection"].ConnectionString;

static public OleDbConnection GetConnection() => new OleDbConnection(connectStr);

}

}

using System;

using System.Collections.Generic;

using System.Data.OleDb;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

namespace CourseApp.Repository

{

public static class Repository<T>

where T: class, new()

{

public static void Add(T item)

{

Type type = typeof(T);

PropertyInfo[] properties = type.GetProperties();

try

{

using (OleDbConnection connection = DB.GetConnection())

{

connection.Open();

string sql = "INSERT INTO [" + type.Name + "] (";

for (int i = 1; i < properties.Length; ++i)

sql += "[" + properties[i].Name + "]" + (i != properties.Length - 1 ? ", " : ")");

sql += " VALUES (";

for (int i = 1; i < properties.Length; ++i)

sql += "?" + (i != properties.Length - 1 ? ", " : ")");

OleDbCommand command = new OleDbCommand(sql, connection);

for (int i = 1; i < properties.Length; ++i)

command.Parameters.AddWithValue(properties[i].Name, properties[i].GetValue(item));

command.ExecuteNonQuery();

}

}

catch (Exception exception)

{

MessageBox.Show("Запрос \"Add()\" в таблице " + type.Name + " не удался.\n" +

exception.Message);

}

}

public static void Remove(T item)

{

Type type = typeof(T);

PropertyInfo[] properties = type.GetProperties();

try

{

using (OleDbConnection connection = DB.GetConnection())

{

connection.Open();

string sql = "DELETE FROM [" + type.Name + "] WHERE (" + properties[0].Name + "=?)";

OleDbCommand command = new OleDbCommand(sql, connection);

command.Parameters.AddWithValue(properties[0].Name, properties[0].GetValue(item));

command.ExecuteNonQuery();

}

}

catch (Exception exception)

{

MessageBox.Show("Запрос \"Remove()\" в таблице " + type.Name + " не удался.\n" +

exception.Message);

}

}

public static void Update(T item)

{

Type type = typeof(T);

PropertyInfo[] properties = type.GetProperties();

try

{

using (OleDbConnection connection = DB.GetConnection())

{

connection.Open();

string sql = "UPDATE [" + type.Name + "] SET ";

for (int i = 1; i < properties.Length; ++i)

sql += "[" + properties[i].Name + "]=?" + (i != properties.Length - 1 ? ", " : "");

sql += " WHERE ([" + properties[0].Name + "]=?)";

OleDbCommand command = new OleDbCommand(sql, connection);

for (int i = 1; i < properties.Length; ++i)

command.Parameters.AddWithValue(properties[i].Name, properties[i].GetValue(item));

command.Parameters.AddWithValue(properties[0].Name, properties[0].GetValue(item));

command.ExecuteNonQuery();

}

}

catch (Exception exception)

{

MessageBox.Show("Запрос \"Update()\" в таблице " + type.Name + " не удался.\n" +

exception.Message);

}

}

public static List<int> GetIds(T item)

{

List<int> result = new List<int>();

Type type = typeof(T);

PropertyInfo[] properties = type.GetProperties();

try

{

using (OleDbConnection connection = DB.GetConnection())

{

connection.Open();

string sql = "SELECT [" + properties[0].Name + "] FROM [" + type.Name + "] WHERE (";

for (int i = 1; i < properties.Length; ++i)

sql += "[" + properties[i].Name + "]=?" + (i != properties.Length - 1 ? " AND " : ")");

OleDbCommand command = new OleDbCommand(sql, connection);

for (int i = 1; i < properties.Length; ++i)

command.Parameters.AddWithValue(properties[i].Name, properties[i].GetValue(item));

OleDbDataReader reader = command.ExecuteReader();

while (reader.Read())

result.Add((int) reader.GetValue(0));

}

}

catch (Exception exception)

{

MessageBox.Show("Запрос \"GetIds()\" в таблице " + type.Name + " не удался.\n" +

exception.Message);

}

return result;

}

public static List<T> GetAll()

{

List<T> result = new List<T>();

Type type = typeof(T);

PropertyInfo[] properties = type.GetProperties();

try

{

using (OleDbConnection connection = DB.GetConnection())

{

connection.Open();

string sql = "SELECT \* FROM [" + type.Name + "]";

OleDbCommand command = new OleDbCommand(sql, connection);

OleDbDataReader reader = command.ExecuteReader();

while(reader.Read())

{

T value = new T();

for (int i = 0; i < properties.Length; ++i)

properties[i].SetValue(value, reader.GetValue(i));

result.Add(value);

}

}

}

catch (Exception exception)

{

MessageBox.Show("Запрос \"GetAll()\" в таблице " + type.Name + " не удался.\n" +

exception.Message);

}

return result;

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class AgentService

{

public static int GetDepartamentIdByAgentId(int agentId)

{

var workers = Repository<Worker>.GetAll();

var agents = Repository<Agent>.GetAll();

var result =

from worker in workers

from agent in agents

where agent.AgentId == agentId

where agent.WorkerId == worker.WorkerId

select worker.DepartamentId;

return result.First();

}

public static Agent GetAgentByUserId(int userId)

{

var agents = Repository<Agent>.GetAll();

var workers = Repository<Worker>.GetAll();

var result =

from agent in agents

from worker in workers

where agent.WorkerId == worker.WorkerId

where worker.UserId == userId

select agent;

return result.First();

}

public static string GetFullNameById(int agentId)

{

var agents = Repository<Agent>.GetAll();

var workers = Repository<Worker>.GetAll();

var result =

from agent in agents

from worker in workers

where agent.WorkerId == worker.WorkerId

where agent.AgentId == agentId

select worker.FullName;

return result.First();

}

public static string GetFullInfoById(int agentId)

{

var agents = Repository<Agent>.GetAll();

var workers = Repository<Worker>.GetAll();

var departaments = Repository<Departament>.GetAll();

var dTypes = Repository<DType>.GetAll();

var filials = Repository<Filial>.GetAll();

var result =

from agent in agents

from worker in workers

from departament in departaments

from dType in dTypes

from filial in filials

where agent.WorkerId == worker.WorkerId

where departament.FilialId == filial.FilialId

where departament.DTypeID == dType.DTypeId

where worker.DepartamentId == departament.DepartamentId

where agent.AgentId == agentId

select new

{

agent.AgentId,

worker.FullName,

worker.MinSalary,

Departament = dType.Title,

FilialCity = filial.City

};

return result.First().ToString();

}

public static bool ContainsWorkerId(int workerId)

{

return Repository<Agent>.GetAll().Any(x => x.WorkerId == workerId);

}

public static int GetSalary(int agentId)

{

var agent = Repository<Agent>.GetAll().First(x => x.AgentId == agentId);

var worker = Repository<Worker>.GetAll().First(x => x.WorkerId == agent.WorkerId);

var contracts = Repository<Contract>.GetAll();

var result =

from contract in contracts

where contract.AgentId == agentId

select contract.Cost;

return result.Sum() + worker.MinSalary;

}

public static IEnumerable<string> GetFullNamesByDepartamentId(int departamentId)

{

var agents = Repository<Agent>.GetAll();

var workers = Repository<Worker>.GetAll();

var result =

from agent in agents

from worker in workers

where agent.WorkerId == worker.WorkerId

where worker.DepartamentId == departamentId

select worker.FullName;

return result;

}

public static Agent GetAgentByFullnameAndDepartamentId(string fullName, int departamentId)

{

var agents = Repository<Agent>.GetAll();

var workers = Repository<Worker>.GetAll();

var result =

from agent in agents

from worker in workers

where agent.WorkerId == worker.WorkerId

where worker.FullName == fullName

where worker.DepartamentId == departamentId

select agent;

return result.First();

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class ClientService

{

public static string GetFullInfoById(int clientId)

{

var clients = Repository<Client>.GetAll();

var result =

from client in clients

where client.ClientId == clientId

select new

{

client.ClientId,

client.FullName,

client.PhoneNumber,

client.Adress

};

return result.First().ToString();

}

public static Client GetClientByUserId(int userId)

{

return Repository<Client>.GetAll().First(x => x.UserId == userId);

}

public static void Add(Client client)

{

Repository<Client>.Add(client);

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class ContractService

{

public static void Add(Contract contract)

{

Repository<Contract>.Add(contract);

}

public static void Update(Contract contract) => Repository<Contract>.Update(contract);

public static void Remove(Contract contract) => Repository<Contract>.Remove(contract);

public static Contract GetContractById(int contractId)

{

return Repository<Contract>.GetAll()

.First(x => x.ContractId == contractId);

}

// by client

public static IEnumerable<object> GetFullInfoByClientId(int clientId)

{

var contracts = Repository<Contract>.GetAll();

var agents = Repository<Agent>.GetAll();

var workers = Repository<Worker>.GetAll();

var services = Repository<Service>.GetAll();

var statuses = Repository<Status>.GetAll();

var result =

from contract in contracts

from agent in agents

from worker in workers

from service in services

from status in statuses

where agent.WorkerId == worker.WorkerId

where contract.AgentId == agent.AgentId

where contract.ServiceId == service.ServiceId

where contract.StatusId == status.StatusId

where contract.ClientId == clientId

select new

{

contract.ContractId,

contract.DateOfConclusion,

ServiceName = service.Title,

AgentName = worker.FullName,

contract.Cost,

contract.InsurancePeriod,

contract.Comment,

Status = status.Title

};

return result.OrderBy(x => x.ContractId);

}

public static IEnumerable<int> GetContractIdByClientId(int clientId)

{

var contracts = Repository<Contract>.GetAll();

var result =

from contract in contracts

where contract.ClientId == clientId

orderby contract.ContractId

select contract.ContractId;

return result;

}

// by agent

public static IEnumerable<object> GetFullInfoByAgentId(int agentId)

{

var contracts = Repository<Contract>.GetAll();

var clients = Repository<Client>.GetAll();

var services = Repository<Service>.GetAll();

var statuses = Repository<Status>.GetAll();

var result =

from contract in contracts

from client in clients

from service in services

from status in statuses

where contract.AgentId == agentId

where contract.ClientId == client.ClientId

where contract.ServiceId == service.ServiceId

where contract.StatusId == status.StatusId

select new

{

contract.ContractId,

contract.DateOfConclusion,

ServiceName = service.Title,

ClientName = client.FullName,

contract.Cost,

contract.InsurancePeriod,

contract.Comment,

Status = status.Title

};

return result.OrderBy(x => x.ContractId);

}

public static List<Contract> GetContractsInProcessingByAgentId(int agentId)

{

return Repository<Contract>.GetAll()

.Where(x => x.AgentId == agentId && x.StatusId == 2)

.ToList();

}

public static List<Contract> GetConfirmedContractsByAgentId(int agentId)

{

return Repository<Contract>.GetAll()

.Where(x => x.AgentId == agentId && x.StatusId == 1)

.ToList();

}

// by manager

public static IEnumerable<object> GetReportByManagerId(int managerId)

{

var contracts = Repository<Contract>.GetAll();

var managers = Repository<Manager>.GetAll();

var agents = Repository<Agent>.GetAll();

var workers = Repository<Worker>.GetAll();

var clients = Repository<Client>.GetAll();

var services = Repository<Service>.GetAll();

var departaments = Repository<Departament>.GetAll();

var dTypes = Repository<DType>.GetAll();

var departamentId = ManagerService.GetDepartamentIdByManagerId(managerId);

var agentInfos =

from agent in agents

from worker in workers

where agent.WorkerId == worker.WorkerId

where worker.DepartamentId == departamentId

select new

{

agent.AgentId,

AgentName = worker.FullName,

};

var result =

(from contract in contracts

where contract.StatusId == 1

from agentInfo in agentInfos

select new

{

AgentId = agentInfo.AgentId,

AgentName = agentInfo.AgentName,

SumOfContracts =

(from agentContract in contracts

where agentContract.AgentId == agentInfo.AgentId

select agentContract.Cost).Sum(),

CountOfContracts =

(from agentContract in contracts

where agentContract.AgentId == agentInfo.AgentId

select agentContract.Cost).Count()

}).Distinct();

return result;

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class DepartamentService

{

public static IEnumerable<string> GetDepartamentTitlesByFilial(Filial filial)

{

var dTypes = Repository<DType>.GetAll();

var departaments = Repository<Departament>.GetAll();

var result =

from dType in dTypes

from departament in departaments

where dType.DTypeId == departament.DTypeID

where departament.FilialId == filial.FilialId

select dType.Title;

return result;

}

public static Departament GetDepartamentByTitleAndFilialId(string title, int filialId)

{

var dTypes = Repository<DType>.GetAll();

var departaments = Repository<Departament>.GetAll();

var result =

from dType in dTypes

from departament in departaments

where dType.DTypeId == departament.DTypeID

where dType.Title == title

where departament.FilialId == filialId

select departament;

return result.First();

}

public static Departament GetDepartamentById(int departamentId)

{

var departaments = Repository<Departament>.GetAll();

var result =

from departament in departaments

where departament.DepartamentId == departamentId

select departament;

return result.First();

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class DTypeService

{

public static DType GetDtypeById(int dTypeId)

{

return Repository<DType>.GetAll()

.Find(x => x.DTypeId == dTypeId);

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class FilialService

{

public static List<string> GetCities()

{

return Repository<Filial>.GetAll().Select(x => x.City).Distinct().ToList();

}

public static Filial GetFilialByCity(string city)

{

return Repository<Filial>.GetAll().First(x => x.City == city);

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class ManagerService

{

public static int GetDepartamentIdByManagerId(int managerId)

{

var workers = Repository<Worker>.GetAll();

var managers = Repository<Manager>.GetAll();

var result =

from worker in workers

from manager in managers

where manager.ManagerId == managerId

where manager.WorkerId == worker.WorkerId

select worker.DepartamentId;

return result.First();

}

public static Manager GetManagerByUserId(int userId)

{

var managers = Repository<Manager>.GetAll();

var workers = Repository<Worker>.GetAll();

var result =

from manager in managers

from worker in workers

where manager.WorkerId == worker.WorkerId

where worker.UserId == userId

select manager;

return result.First();

}

public static int GetSalary(int managerId)

{

var manager = Repository<Manager>.GetAll().First(x => x.ManagerId == managerId);

var managerWorker = Repository<Worker>.GetAll().First(x => x.WorkerId == manager.WorkerId);

int departamentId = GetDepartamentIdByManagerId(managerId);

var contracts = Repository<Contract>.GetAll();

var workers = Repository<Worker>.GetAll();

var agents = Repository<Agent>.GetAll();

var result =

(from contract in contracts

from agent in agents

from worker in workers

where worker.WorkerId == agent.WorkerId

where contract.AgentId == agent.AgentId

where worker.DepartamentId == departamentId

select agent.AgentId).Distinct();

return (int)(result.Select(x => AgentService.GetSalary(x)).Sum() \* 0.3 + managerWorker.MinSalary);

}

public static string GetFullInfoById(int managerId)

{

var managers = Repository<Manager>.GetAll();

var workers = Repository<Worker>.GetAll();

var departaments = Repository<Departament>.GetAll();

var dTypes = Repository<DType>.GetAll();

var filials = Repository<Filial>.GetAll();

var result =

from manager in managers

from worker in workers

from departament in departaments

from dType in dTypes

from filial in filials

where manager.WorkerId == worker.WorkerId

where departament.FilialId == filial.FilialId

where departament.DTypeID == dType.DTypeId

where worker.DepartamentId == departament.DepartamentId

where manager.ManagerId == managerId

select new

{

manager.ManagerId,

worker.FullName,

worker.MinSalary,

Departament = dType.Title,

FilialCity = filial.City

};

return result.First().ToString();

}

public static bool ContainsWorkerId(int workerId)

{

return Repository<Manager>.GetAll().Any(x => x.WorkerId == workerId);

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class ServiceService

{

public static List<Service> GetServices()

{

return Repository<Service>.GetAll()

.OrderBy(x => x.ServiceId).ToList();

}

public static List<string> GetTitles()

{

return Repository<Service>.GetAll().Select(x => x.Title).ToList();

}

public static IEnumerable<Service> GetServicesByAgentId(int agentId)

{

var agents = Repository<Agent>.GetAll();

var workers = Repository<Worker>.GetAll();

var departaments = Repository<Departament>.GetAll();

var dTypes = Repository<DType>.GetAll();

var services = Repository<Service>.GetAll();

var result =

from agent in agents

from worker in workers

from service in services

from departament in departaments

from dType in dTypes

where agent.AgentId == agentId

where agent.WorkerId == worker.WorkerId

where worker.DepartamentId == departament.DepartamentId

where dType.DTypeId == departament.DTypeID

where dType.DTypeId == service.DTypeId

select service;

return result;

}

public static void Update(Service service) => Repository<Service>.Update(service);

public static IEnumerable<string> GetNamesByDtypeId(int dTypeId)

{

return Repository<Service>.GetAll()

.Where(x => x.DTypeId == dTypeId)

.Select(x => x.Title);

}

public static Service GetServiceByTitleAndDTypeId(string title, int dTypeId)

{

return Repository<Service>.GetAll()

.First(x => x.Title == title && x.DTypeId == dTypeId);

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class UserService

{

public static bool Contains(User user)

{

return Repository<User>.GetAll().Any(x => x.Login == user.Login && x.Password == user.Password);

}

public static bool ContainsLogin(string login)

{

return Repository<User>.GetAll().Any(x => x.Login == login);

}

public static int GetId(User user)

{

return Repository<User>.GetAll().First(x => x.Login == user.Login && x.Password == user.Password).UserId;

}

public static Role GetRole(int userId)

{

if (WorkerService.ContainsUserId(userId)) // \_\_worker

{

int workerId = WorkerService.GetWorkerIdByUserId(userId);

if (AgentService.ContainsWorkerId(workerId))

return Role.Agent; // agent

else

return Role.Manager; // manager

} else // client

{

return Role.Client;

}

}

public static void Add(User user)

{

Repository<User>.Add(user);

}

}

}

using CourseApp.Classes;

using CourseApp.Repository;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CourseApp.Services

{

static class WorkerService

{

public static void Update(Worker worker) => Repository<Worker>.Update(worker);

public static List<Worker> GetWorkersByDepartamentId(int departamentId)

{

var workers = Repository<Worker>.GetAll();

var result =

from worker in workers

where worker.DepartamentId == departamentId

select worker;

return result.ToList();

}

public static bool ContainsUserId(int userId)

{

return Repository<Worker>.GetAll().Any(x => x.UserId == userId);

}

public static int GetWorkerIdByUserId(int userId)

{

return Repository<Worker>.GetAll().First(x => x.UserId == userId).WorkerId;

}

}

}