**ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ** **ГОСУДАРСТВЕННОЕ-БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЯ ГОРОДА МОСКВЫ**

**МОСКОВСКИЙ АВТОМОБИЛЬНО – ДОРОЖНЫЙ** **КОЛЛЕДЖ**

**ИМ. А. А. НИКОЛАЕВА**

ВКР допущена к защите

Заместитель директора

\_\_\_\_\_\_\_\_\_Т.В. Гончарук

«\_\_\_\_» \_\_\_\_\_\_\_\_\_2022г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**Тема:** **Разработка комплекса программного обеспечения для компании по предоставлению услуг автострахования**

Дипломник \_\_\_\_\_\_\_\_\_\_\_\_ Шебанов А.А.

Руководитель\_\_\_\_\_\_\_\_\_\_\_ Сянина Т.Г.

Рецензент \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Касаткина М.А.

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2022 г.

Москва

2022

**ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ И НАУКИ ГОРОДА МОСКВЫ ГОСУДАРСТВЕННОЕ-БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЯ ГОРОДА МОСКВЫ**

**МОСКОВСКИЙ АВТОМОБИЛЬНО – ДОРОЖНЫЙ КОЛЛЕДЖ**

**ИМ. А. А. НИКОЛАЕВА**

УТВЕРЖДАЮ

Заместитель директора

\_\_\_\_\_\_\_\_\_Т.В. Гончарук

«\_\_\_\_» \_\_\_\_\_\_\_\_\_2022 г.

**ЗАДАНИЕ**

к выпускной квалификационной работе

студента очной формы обучения, группы 4ИП

специальности 09.02.07 «Информационные системы и программирование»

Фамилия, имя, отчество Шебанов Антон Андреевич

Тема выпускной квалификационной работы Разработка комплекса программного обеспечения для компании по предоставлению услуг автострахования

1. Постановка задачи:

Разработать программное обеспечение с дружественным и интуитивно понятным интерфейсом для повышения оперативности работы и улучшения контроля учета автострахования.

Функционал программы должен обладать возможностью хранения данных о водителях, клиентах, автомобилях и ОСАГО.

1. Исходные данные:

Информация о ОСАГО, водителях, машинах, работниках и коэффициентах.

1. Перечень вопросов, подлежащих разработке в выпускной квалификационной работе:
2. Cформулировать постановку задачи и исследовать предметную область;
3. Проанализировать существующие программные обеспечения по автоматизации деятельности компании по автострахованию;
4. Спроектировать концептуальную модель данных;
5. Спроектировать информационно-логическую модель данных;
6. Проанализировать и выбрать OC, СУБД и язык программирования;
7. Спроектировать физическую модель данных;
8. Разработать интерфейс комплекса программного обеспечения;
9. Определить технические и программные средства реализации проекта;
10. Проверить работоспособность комплекса программного обеспечения средствами тестирования;
11. Разработать руководство пользователя. ­

Задание принял к исполнению «\_\_\_\_» 2022 г.

Студент А.А.Шебанов

И.О. Фамилия, подпись

Срок сдачи «\_\_\_\_» 2022 г.

Руководитель выпускной квалификационной работы Т.Г. Сянина

И.О. Фамилия, подпись

Задание рассмотрено на заседании цикловой комиссии «\_\_\_\_» 2022 г.

Председатель выпускающей комиссии / М.С.Яковлев

подпись И.О. Фамилия

**СОДЕРЖАНИЕ**

[СПИСОК СОКРАЩЕНИЙ 5](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315570)

[ВВЕДЕНИЕ 6](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315571)

[1. ТЕОРИТИЧЕСКИЙ РАЗДЕЛ 8](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315572)

[1.1 Основные аспекты разработки комплекса ПО 8](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315573)

[1.2 Анализ разработанных программных обеспечений в области автострахования 18](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315574)

[2. РАЗДЕЛ ПРОЕКТИРОВАНИЯ 21](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315575)

[2.1 Постановка задачи 21](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315576)

[2.2 Описание предметной области 21](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315577)

[2.3 Анализ, обоснование и выбор средств реализации комплекса программного обеспечения (ОС, СУБД, ЯП) 21](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315578)

[2.4 Технические требования к разрабатываемому программному обеспечению 38](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315579)

[2.4.1 Проектирование диаграммы прецедентов, концепт модели 39](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315580)

[2.4.2 Проектирование инфологической модели и словаря данных 39](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315581)

[2.4.3 Физическая модель 44](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315582)

[2.5 Разработка интерфейса комплекса ПО 45](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315583)

[3. ЭКСПЕРЕМЕНТАЛЬНЫЙ РАЗДЕЛ 54](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315584)

[3.1 Тестирование комплекса ПО 57](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315585)

[3.2 Отладка разработанного комплекса ПО 60](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315586)

[ЗАКЛЮЧЕНИЕ 61](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315587)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 62](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315588)

[ПРИЛОЖЕНИЕ 1 64](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315589)

[ПРИЛОЖЕНИЕ 2 71](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315590)

[ПРИЛОЖЕНИЕ 3 75](file:///C:\Users\Hulk75\Downloads\KR.docx#_Toc67315591)

СПИСОК СОКРАЩЕНИЙ

HTML - HyperText Markup Language;

JSON - JavaScript Object Notation;

SQL - Structured Query Language;

TFS - Team Foundation Server;

UML - Unified Modeling Language;

WPF - Windows Presentation Foundation;

ЖЦ - жизненный цикл;

ИС - информационные системы;

ОС - операционная система;

ОСАГО - обязательное страхование автогражданской ответственности;

ПК - персональный компьютер;

ПО - программное обеспечение;

СУБД - система управления базами данных;

ФМД - физическая модель данных.

ВВЕДЕНИЕ

Развитие информационных технологий привело к возможности цифровизации страхового рынка. Цифровые технологии применяются к процессу продаж страховых продуктов в первую очередь в массовых видах страхования, где страховщики предлагают, как правило, однородные услуги, отличающиеся ценой и сервисной составляющей. При этом происходит образование маркетплейсов и агрегаторов, информирующих потребителей о предлагаемых страховых услугах и позволяющих сравнивать цены на них.

Актуальность темы выпускной квалификационной работы заключается в значимости страховой деятельности для национальной экономики, финансовой системы государства и защиты законных интересов граждан и юридических лиц, а также модули учета и редактирования данных позволят значительно облегчить выполняемые сотрудниками задачи, а также в повышении уровня менеджмента.

Объектом исследования в дипломной работе является компания, предоставляющая услуги по автострахованию.

Предметом исследования в дипломной работе является автоматизация работы персонала компании по автострахованию.

Основной целью дипломной работы является разработка комплекса программного обеспечения для компании по автострахованию, позволяющей автоматизировать деятельность работников организации.

Для достижения постановленной цели необходимо решить следующие задачи:

* Сформулировать постановку задачи и исследовать предметную область;
* Проанализировать существующие программные обеспечения по автоматизации деятельности компании по автострахованию;
* Спроектировать концептуальную модель данных;
* Спроектировать информационно-логическую модель данных;
* Проанализировать и выбрать OC, СУБД и язык программирования;
* Спроектировать физическую модель;
* Разработать интерфейс комплекса программного обеспечения;
* Определить технические и программные средства реализации проекта;
* Проверить работоспособность программного обеспечения средствами тестирования;
* Разработать руководство-пользователя.

# **1. ТЕОРИТИЧЕСКИЙ РАЗДЕЛ**

## **1.1 Основные аспекты разработки комплекса ПО**

У программного обеспечения, как у живого существа есть свой ЖЦ. Для природы будет верно определение жизненного цикла как закономерную смену всех поколений, характерных для данного вида живых организмов, а для программного обеспечения будет верно следующее определение жизненного цикла как стадии, которые проходит программный продукт от появления идеи до ее реализации в коде, имплементации в бизнес и последующей поддержки. Модели жизненного цикла во многом предопределяют и методологии разработки ПО.

Обычно к этапам жизненного цикла относят:

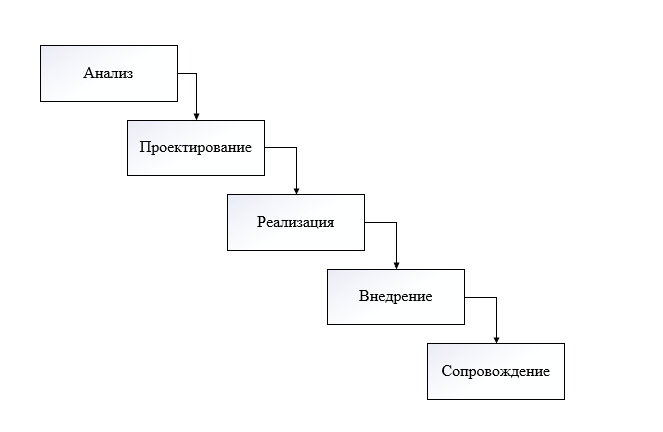
* Анализ требований;
* Проектирование;
* Программирование;
* Тестирование и отладку;
* Эксплуатацию, сопровождение и поддержку.

Существует некая вариативность в прохождении этапов ЖЦ во время разработки и внедрения продукта на рынок. Для каждого продукта это происходит по-своему, но, чтобы процессом как-то управлять были сформулированы **модели жизненного цикла ПО**– упрощенное и обобщенное представление о том, как развивается продукт. Существует несколько моделей жизненного цикла. Давайте посмотрим на них подробнее и разберемся, что в них общего, а что отличается.

Первой рассмотренной моделью будет каскадная.

Самой старой и известной моделью построения многоуровневого процесса разработки является каскадная модель.

Каскадная модель, изображенная на рисунке 1.1, предусматривает последовательную организацию работ. Демонстрируется классический подход к разработке различных систем из любой ПО. В ранее существовавших ИС каждое приложение представляло собой единое целое. Для разработки такого типа приложений применялся каскадный способ. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем. Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.



**Рисунок 1.1** – Каскадная модель ЖЦ

Модель делиться на этапы:

* этап, анализ – исследование проблемы, четкая формулировка требований заказчика. Результат – техническое задание, согласованное со всеми заинтересованными сторонами;
* этап, проектирование – разработка проектных решений, удовлетворяющих всем требованиям технического задания. Результат – комплект проектной документации, содержащей все необходимые данные для реализации проекта;
* этап, реализация - программирование в соответствии с проектными решениями, полученными на предыдущем этапе. Методы, используемые для реализации, не имеют принципиального значения. На этом этапе также производится проверка ПО на предмет соответствия требованиям технического задания, выявление скрытых недостатков, проявляющихся в реальных условиях работы ИС. Результат – готовый программный продукт;
* этап, внедрение - сдача готового проекта. Главная задача – убедить заказчика, что все его требования реализованы в полной мере;
* этап, сопровождение - в ходе сопровождения в программу вносятся изменения с тем, чтобы исправить обнаруженные в процессе использования дефекты и недоработки, а также для добавления новой функциональности, с целью повысить удобство использования и применимость ПО.

Каскадная модель имеет следующие преимущества:

* процесс разработки выполняется поэтапно;
* ее структурой может руководствоваться даже слабо подготовленный в техническом плане или - неопытный персонал;
* она способствует осуществлению строгого контроля менеджмента проекта;
* каждую стадию могут выполнять независимые команды (все документировано);
* позволяет достаточно точно планировать сроки и затраты;
* на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
* выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Недостатки каскадной модели связаны с тем, что реальный процесс создания ПО никогда не укладывается в такую жесткую схему. Недостатки каскадной модели особо остро проявляются в случае, когда трудно (или невозможно) сформулировать требования или требования могут меняться в процессе выполнения проекта. В этом случае разработка комплекса ПО имеет принципиально циклический характер. Основные недостатки:

* существенная задержка получения результатов;
* ошибки и недоработки на любом этапе выясняются, как правило, на последующих этапах работы, что приводит к необходимости возврата на предыдущие стадии;
* сложность распараллеливания работ по проекту;
* чрезмерная информационная перенасыщенность каждого из этапов;
* сложность управления проектом;
* высокий уровень риска и ненадежность инвестиций.

После перейдем к рассмотрению итеративной модели, показанной на рисунке 1.2.



**Рисунок 1.2** – Итеративная модель ЖЦ

Итеративная модель не предполагает полного объема требований для начала работ над продуктом. Разработка программы может начинаться с требований к части функционала, которые могут впоследствии дополняться и изменяться.  Процесс повторяется, обеспечивая создание новой версии продукта для каждого цикла.

В несколько упрощенном виде, итеративная модель состоит из четырех основных стадий, которые повторяются в каждой из итераций:

* определение и анализ требований;
* дизайн и проектирование – согласно требованиями. Причем дизайн может как разрабатываться отдельно для данной функциональности, так и дополнять уже существующий;
* разработка и тестирование – кодирование, интеграция и тестирование нового компонента;
* фаза ревью – оценка, пересмотр текущих требований и предложения дополнений к ним.

По результатам каждой итерации принимается решение – будут ли использованы ее результаты для дополнения существующей функциональности в качестве входной точки для начала следующей итерации (т.н. инкрементальное прототипирование). В конечном итоге, достигается точка, в которой все требования были воплощены в продукте – происходит релиз.

В математических терминах, итеративная модель представляет реализацию методики последовательной аппроксимации.

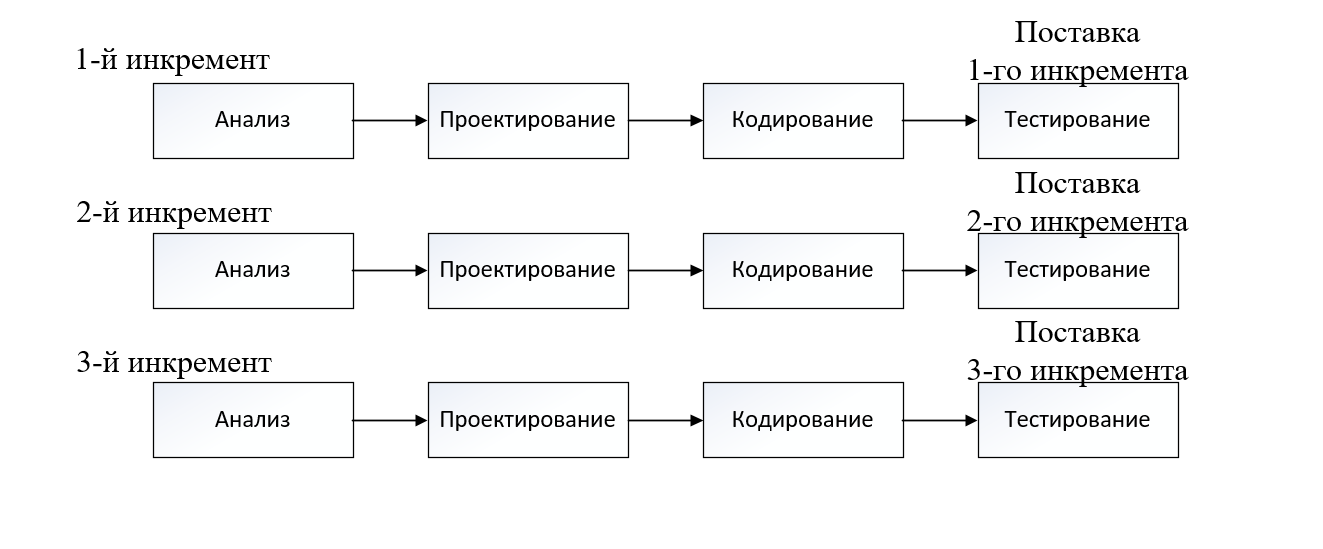
Ключ к успешному использованию этой модели – строгая валидация требований и тщательная верификация разрабатываемой функциональности в каждой из итераций.

Основные стадии процесса разработки в итеративной модели фактически повторяют модель водопада. В каждой итерации создается программное обеспечение, требующее тестирования на всех уровнях.

Плюсы итеративной модели:

* раннее создание работающего ПО;
* гибкость – готовность к изменению требований на любом этапе разработки;
* каждая итерация – маленький этап, для которого тестирование и анализ рисков обеспечить проще, чем для всего жизненного цикла продукта.
* Минусы итеративной модели:
* каждая фаза – самостоятельна, отдельные итерации не накладываются;
* могут возникнуть проблемы с реализацией общей архитектуры системы, поскольку не все требования известны к началу проектирования.

Перейдем к рассмотрению инкрементной модели ЖЦ, показанной на рисунке 1.3.

****

**Рисунок 1.3** – Инкрементная модель ЖЦ

Инкрементная модель — это метод, в котором проект проектируется, реализуется и тестируется инкрементно (то есть каждый раз с небольшими добавлениями) до самого окончания разработки. Это включает в себя как разработку, так и дальнейшую поддержку продукта. Он считается законченным в то время, когда удовлетворяет всем требованиям. Модель объединяет элементы каскадной модели с прототипированием.

Проект можно разложить на несколько составляющих, каждая из которых спроектирована и построена независимо от другой (билд). Каждый такой компонент поставляется клиенту по мере готовности, что позволяет сразу начать использовать продукт и избежать длительной разработки. Также это стимулирует большие инвестиционные затраты, но сокращает время ожидания результата. Модель помогает «сгладить углы» вместо того, чтобы выкатывать пользователю совершенно новую систему разом.

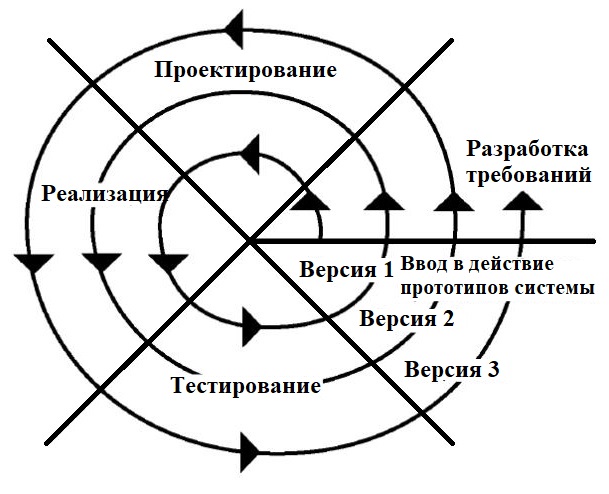
Преимущества инкрементной модели:

* не требуется заранее тратить средства, необходимые для разработки всего проекта, поскольку сначала выполняется разработка и реализация основной функции или функции из группы высокого риска;
* в результате выполнения каждого инкремента получается функциональный продукт;
* правило по принципу "разделяй и властвуй" позволяет разбить возникшую проблему на управляемые части;
* в процессе разработки можно ограничить количество персонала;
* возможность начать построение следующей версии проекта на переходном этапе предыдущей версии сглаживает изменения, вызванные сменой персонала;
* в конце каждой инкрементной поставки существует возможность пересмотреть риски, связанные с затратами и соблюдением установленного графика;
* ускоряется начальный график поставки, что позволяет соответствовать возросшим требованиям рынка;
* потребности клиента лучше поддаются управлению, поскольку время разработки каждого инкремента очень незначительно;
* заказчики могут распознавать самые важные и полезные функциональные возможности продукта на более ранних этапах разработки;
* требования стабилизируются (посредством включения в процесс пользователей) на момент создания определенного инкремента;
* улучшается понимание требований для более поздних инкрементов.

Недостатки инкрементной модели:

* в модели не предусмотрены итерации в рамках каждого инкремента;
* определение полной функциональной системы должно осуществляться в начале жизненного цикла, чтобы обеспечить определение инкрементов;
* поскольку создание некоторых модулей будет завершено значительно раньше других, возникает необходимость в четко определенных интерфейсах;
* формальный критический анализ и проверку намного труднее выполнить для инкрементов, чем для системы в целом;
* для модели необходимы хорошее планирование и проектирование;
* каждый новый билд должен быть интегрирован с предыдущим во всех системах.

Далее перейдем к рассмотрению спиральной модели ЖЦ, показанной на рисунке 1.4.



**Рисунок 1.4 –** Спиральная модель ЖЦ

Схема работы спиральной модели выглядит следующим образом. Разработка вариантов продукта представляется как набор циклов раскручивающейся спирали. Каждому циклу спирали соответствует такое же количество стадий, как и в модели каскадного процесса. При этом начальные стадии, связанные с анализом и планированием, представлены более подробно с добавлением новых элементов. В каждом цикле выделяются четыре базовые фазы:

* определение целей, альтернативных вариантов и ограничений;
* оценка альтернативных вариантов, идентификация и разрешение рисков;
* разработка продукта следующего уровня;
* планирование следующей фазы.

«Раскручивание» проекта начинается с анализа общей постановки задачи на разработку комплекса ПО. Здесь на первой фазе определяются общие цели, устанавливаются предварительные ограничения, определяются возможные альтернативы подходов к решению задачи. Далее проводится оценка подходов, устанавливаются их риски. На шаге разработки создается концепция продукта и путей его создания.

Следующий цикл - разработка проекта - начинается с планирования разработки. На фазе определения целей устанавливаются ограничения проекта (по срокам, объему финансирования, ресурсам и т.д.), определяются альтернативы проектирования, связанные с альтернативами требований, применяемыми технологиями проектирования, привлечением субподрядчиков. На фазе оценки альтернатив устанавливаются риски вариантов и делается выбор варианта для дальнейшей реализации. На фазе разработки выполняется проектирование и создается демоверсия, отражающая основные проектные решения.

Следующий цикл – реализация комплекса ПО - также начинается с планирования. Альтернативными вариантами реализации могут быть применяемые технологии реализации, привлекаемые ресурсы. Оценка альтернатив и связанных с ними рисков на этом цикле определяется степенью «отработанности» технологий и «качеством» имеющихся ресурсов. Фаза разработки выполняется по каскадной модели с выходом – действующим вариантом (прототипом) продукта.

Преимущества:

* существенно упрощается внесение изменений в проект при изменении   
  требований заказчика;
* отдельные элементы ПО интегрируются в единое целое постепенно, причем, процесс интеграции производится фактически непрерывно;
* уменьшение уровня рисков, т.к. риски обнаруживаются именно в процессе интеграции. Риск максимален в начале разработки проекта, по мере продвижения он уменьшается;
* Гибкость в управлении проектом. Например, можно сократить сроки разработки за счет уменьшения функциональности системы или использовать в качестве составных частей вместо своих разработок продукцию других фирм;
* Итерационный подход упрощает повторное использование компонентов. Анализ проекта после проведения нескольких начальных итераций позволяет выявить общие, многократно используемые компоненты, которые на последующих итерациях будут совершенствоваться;
* Спиральная модель позволяет получить более надежную и устойчивую систему, т.к. ошибки и слабые места обнаруживаются и исправляются на каждой итерации.

Недостатки:

* сложность анализа и оценки рисков при выборе вариантов;
* сложность поддержания версий продукта (хранение версий, возврат к ранним версиям, комбинация версий);
* сложность оценки точки перехода на следующий цикл;
* бесконечность модели – на каждом витке заказчик может выдвигать новые требования, которые приводят к необходимости следующего цикла разработки.

Проведя анализ моделей жизненного цикла, была выбрана инкрементная модель по следующим причине, возможности разбить программу на части и поэтапно разрабатывать их.

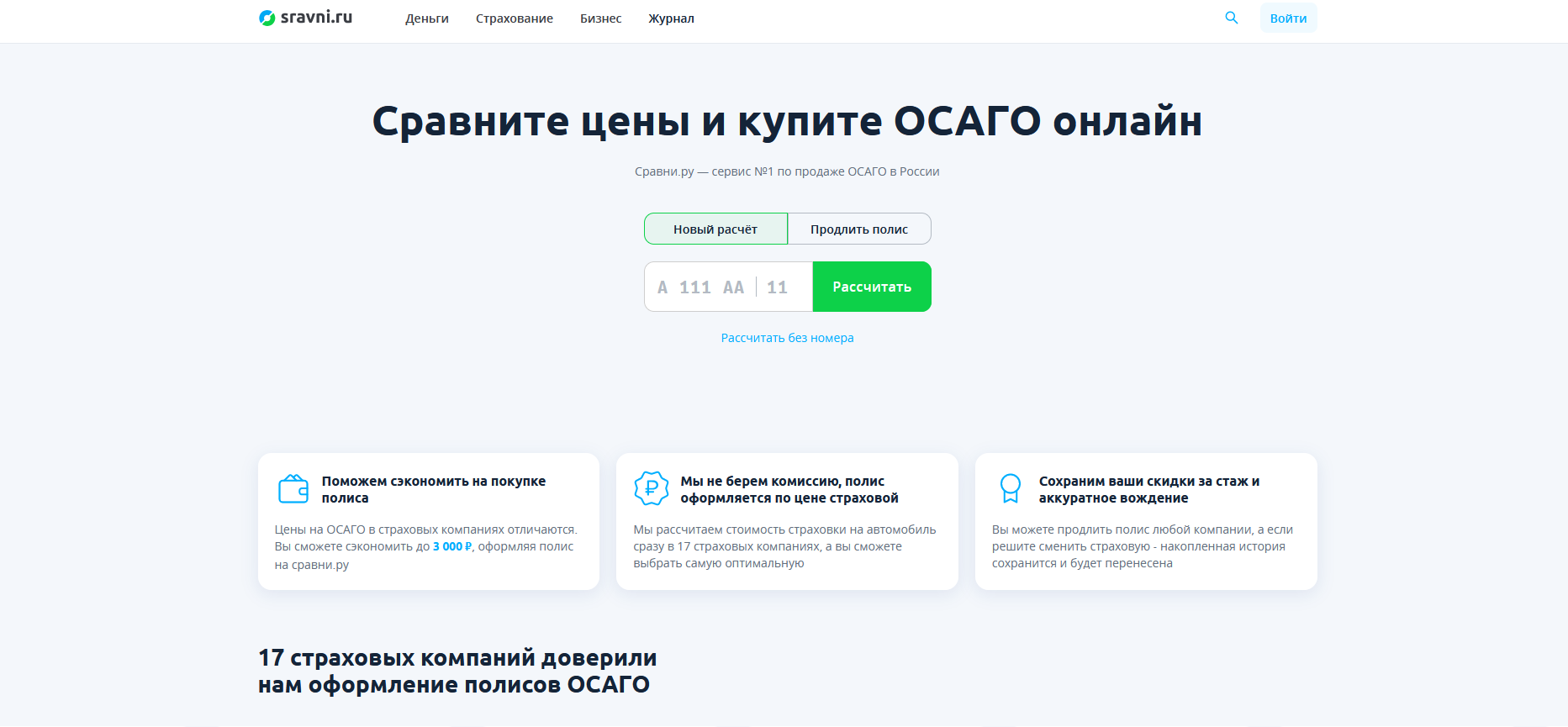
В этом разделе были рассмотрены жизненные циклы и проведен анализ моделей жизненных циклов.

В следующем разделе будет проведен анализ готовой продукции на рынке.

1.2 Анализ разработанных программных обеспечений в области автострахования

На рынке программных обеспечений для осуществления автострахования существуют решения множество решений по данному направлению. Анализ поможет понять общие тенденции в подобных ПО.

Первым для анализа будет сайт: https://www.sravni.ru/osago. Sravni.ru это сервис сравнения банковских услуг подбора кредита, ОСАГО и отзывов страховых компаний. Пример интерфейса сайта показан на рисунке 1.5.



*Рисунок 1.5 – Сайт по автострахованию «sravni.ru»*

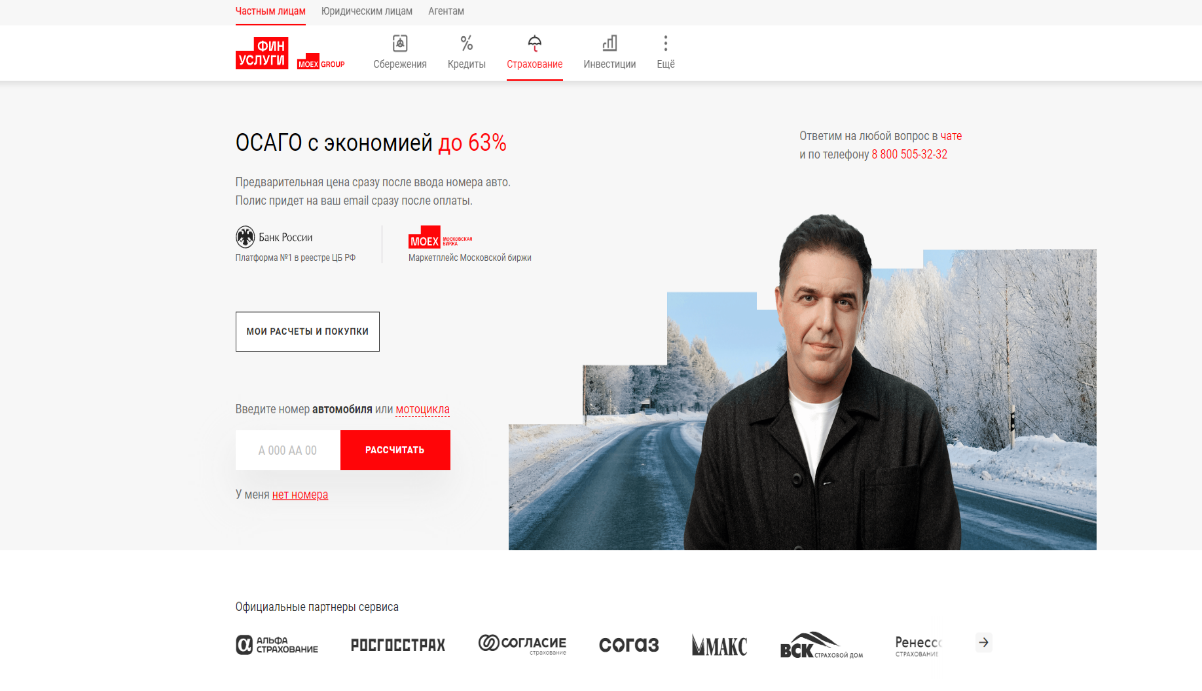
Плюсы данного сайта:

* Широкий выбор страховых компаний;
* Прост в понимании;
* Есть удобный калькулятор;
* Возможность оформить страховой случай.

Минусы:

* Обязательная регистрация;
* Отсутствует просмотр статус страховки.

Вторым сайтом для анализа будет https://osago.finuslugi.ru. Finuslugi.ru финансово-страховой агрегатор по вкладам и страхованию. Пример интерфейса сайта показан на рисунке 1.6.



*Рисунок 1.6 – Сайт по автострахованию «finuslugi.ru»*

Плюсы данного сайта:

* Широкий выбор страховых компаний;
* Прост в понимании;
* Есть удобный калькулятор;
* Гарантия подлинности.

Минусы:

* Обязательная регистрация;
* Необходимость аккаунта в Госуслугах для регистрации;
* Отсутствует просмотр статус страховки.

При анализе программных обеспечений в области автострахования, были выявлены особенные черты, которые могут ускорить обслуживание клиентов в области автострахования.

По итогам анализа было принято решение разработать комплекс программного обеспечения для ПК, которая смогла бы заметно упростить оформление страховки, а также для удобной работы с данными.

Таким образом в данной работе были изучены основные этапы разработки программного обеспечения, а именно основная цель проектирования организация данных и связи между ними. А также исследованы аналоги в сфере автоматизации автострахования.

# **2. РАЗДЕЛ ПРОЕКТИРОВАНИЯ**

2.1 Постановка задачи

Основной задачей дипломной работы является разработать комплекс ПО для автострахования, с целью автоматизации труда работников.

База данных будет включать в себя информацию о сотрудниках, ОСАГО, водителях, автомобилях.

Готовый комплекс программного обеспечения должен обладать интуитивно понятным и удобным интерфейсом.

Разрабатываемый комплекс ПО должен соответствовать следующим требованиям:

* ограничение доступа в зависимости от роли работника;
* просмотр удаление, добавление, изменение записей в таблицах;
* формирование отчетов;
* формирование статистики по разным параметрам;
* интуитивный доступ к различным данным.

К комплексу программного обеспечения должно прилагаться руководство пользователя, которое позволит подробно ознакомится с ее возможностями и правильность использования.

2.2 Описание предметной области

Основной целью дипломной работы является разработка комплекса ПО для учета работы компании по автострахованию транспортных средств, позволяющей автоматизировать деятельность организации предоставляющей услуги по автострахованию транспортного средств.

Директор будет иметь возможность просмотра всех таблиц. В его возможности будет входить формирование отчетов и статистики по разным параметрам.

Администратор будет иметь возможность управления всеми учетными записями БД. В его функционал входит добавление, изменение и удаление данных.

Менеджер по ОСАГО сможет управлять записями в таблицах ОСАГО, водители, автомобили. В его функционал входит добавление и изменение данных.

Работа с клиентами будет происходить через веб-приложение: потенциальный клиент будет делать расчет и оформление ОСАГО, при его запросе данные добавляются в таблицу.

В данном разделе была рассмотрена предметная область, далее перейдем к обоснованию и выбору средств реализации ПО.

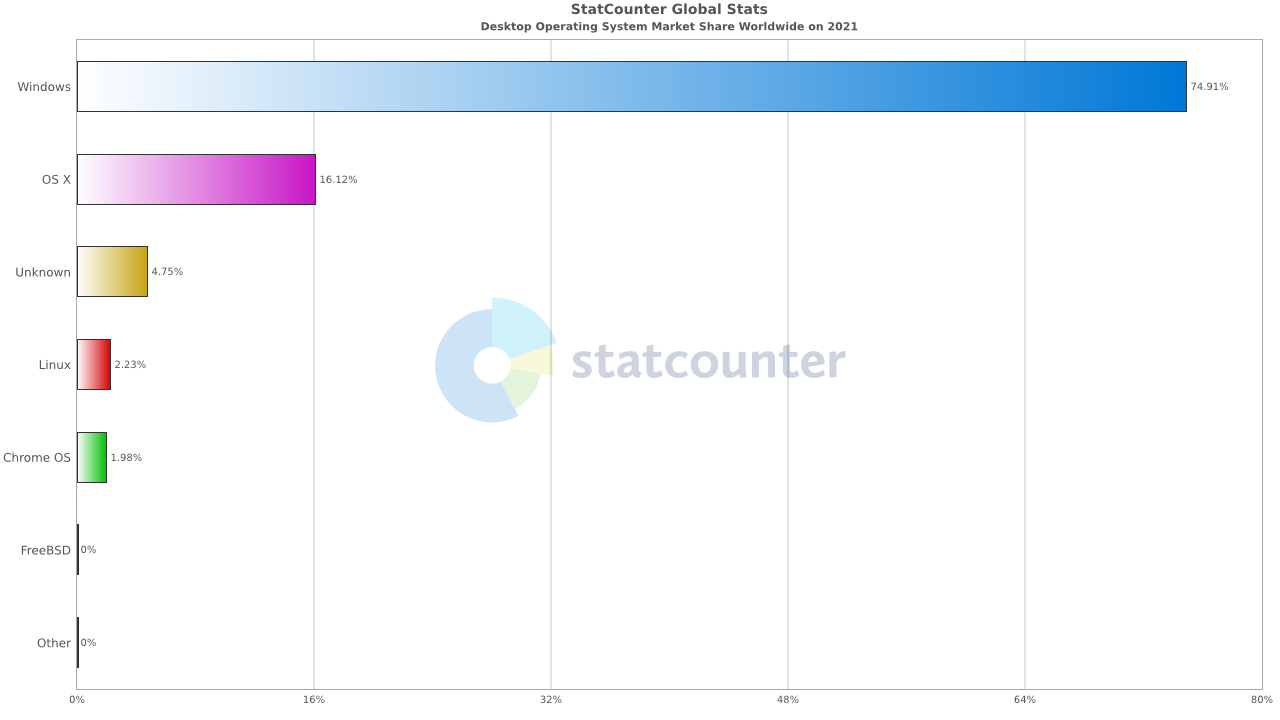
2.3 Анализ, обоснование и выбор средств реализации комплекса программного обеспечения (ОС, СУБД, ЯП)

Первым будет рассмотрена и выбрана операционная система.

Сначала узнаем, что такое ОС.

Операционная система - комплекс взаимосвязанных программ, предназначенных для управления ресурсами компьютера и организации взаимодействия с пользователем.

Рассмотрим популярность ОС на рынке по рисунку 2.1 с сайта StatCounter[[1]](#footnote-1).



**Рисунок 2.1 –** график популярности ОС

На рисунке 1.5 видно, что лидирующие позиции занимают системы Windows 10, OS X или MacOS 10.13 High Sierra, Linux 4.20.

Далее рассмотрим плюсы и минусы каждой ОС.

Первой будет рассмотрена ОС Windows.

Windows 10 - группа семейств коммерческих операционных систем корпорации Microsoft, ориентированных на управление с помощью графического интерфейса. MS-DOS - является прародителем Windows. Изначально, Windows был лишь графическим интерфейсом для MS-DOS.

Плюсы:

* Проста в использовании. Интуитивно понятный интерфейс дает возможность пользователю быстро разобраться в ОС;
* Не нужны знания программирования для использования;
* Большая часть оборудования и софта разрабатываются именно под Windows;
* Легкая настройка ОС под свои нужды;
* Легко восстановить утерянную информацию с флешки или жёсткого диска;
* Большое сообщество пользователей, что позволяет найти ответы на вопросы по ОС в кратчайшие сроки.

Минусы:

* Бесполезный и скудный набор программ в самой ОС. Все идет со стороннего софта и ОС нужна лишь, как платформа для них;
* Слишком много вирусов. Именно из-за массовости данной операционной системы, хакерами пишутся очень много вредоносного программного обеспечения в большом количестве именно под Windows;
* Если не чистить ОС вовремя от различных временных файлов, и лишних программ, Windows начинает заметно медленнее работать;
* Следующим будет рассмотрена система MacOS.

MacOS 10.13 High Sierra - проприетарная операционная система от компании Apple. В MacOS используется ядро XNU, основанное на микроядре Mach и содержащее программный код, разработанный компанией Apple, а также код из ОС NeXTSTEP и FreeBSD.

Плюсы:

* Стабильная, красивая, лаконичная и продуманная в сравнении с другими ОС;
* Установка программного обеспечения упрощена до максимума — обычно достаточно перенести программу с носителя в список программ;
* Мало вирусов, а те, что есть ограничены в возможностях. Если появляется информация об уязвимостях — они быстро перекрываются исправлениями;
* Есть встроенное офисное программное обеспечение;

Минусы:

* Ограниченность системы. Т.к. в системе многое упрощено — нет гибкости в настройках;
* Проблемы с поддержкой оборудования. Не все оборудование может быть совместимо. Могут быть сложности с установкой драйверов;
* Иногда установка программного обеспечения выходит за рамки перенесения значка в список программ. Могут быть скрипты с запуском и работой которых могут быть проблемы;
* Для записи на диски с NTFS нужно ставить дополнительные драйвера;
* Для установки на не родное железо придется потратить много времени. Понадобится переустановка чтобы установить новую версию операционной системы.

Последняя рассмотренная ОС будет Linux.

Linux 4.20 - семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU, и, возможно, другие компоненты. Как и ядро Linux, системы на его основе, как правило, создаются и распространяются в соответствии с моделью разработки свободного и открытого программного обеспечения.

Плюсы:

* Распространяется бесплатно;
* Надежная и стабильная в своей работе ОС;
* Вирусов почти нет. Антивирус не нужен. Можно поставить бесплатный ClamAV для самоуспокоения и периодического сканирования файловой системы;
* Система гибка в настройке, а в сети много информации как, что настраивается;
* Занимает 15 Гб на диске и не разрастается до фантастических размеров с течением времени в сравнении с Windows.

Минусы:

* Любое изменение системных настроек несет риск сбоя. Нужно четко понимать, что делаете. Настроенное ранее может слетать после обновления ядра. Если настроенное не трогать — будет работать очень стабильно;
* Набор программ более ограничен в сравнении с конкурентами. По-умолчанию установка ПО производится преимущественно из репозиториев с ограниченным набором программ;
* Есть проблемы с совместимостью оборудования. Иногда бывают проблемы с установкой драйверов, а когда все вроде установлено бывает, что работает не корректно.

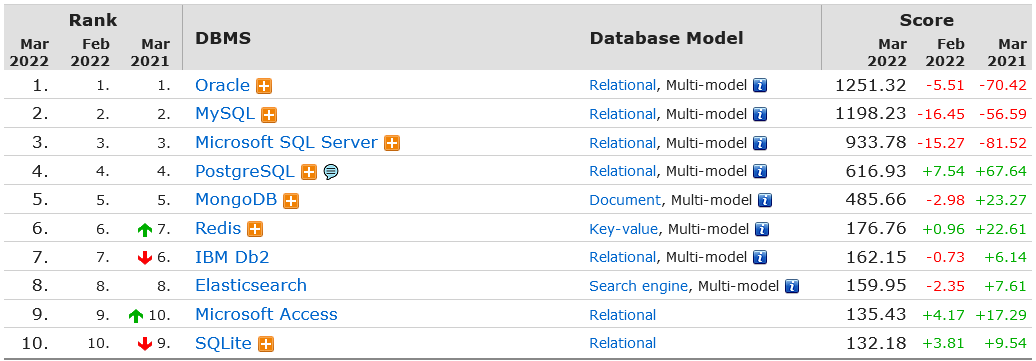
В итоге была выбрана ОС Windows 10, по причине популярности и удобности в использовании, что перекрывает возможные минусы ОС.

После выбора операционной системы, приступим к выбору СУБД.

Выбор СУБД стоит начать с понятия того чем является СУБД.

СУБД или система управления базами данных - комплекс программ, позволяющих создать базу данных (БД) и манипулировать данными. Система обеспечивает безопасность, надёжность хранения и целостность данных, а также предоставляет средства для администрирования БД.

На рисунке Рисунок 2.2 показан актуальный график популярности СУБД, источник сайт DB-engines[[2]](#footnote-2).



**Рисунок 2.2** – график популярности СУБД

По Рисунок 2.2 видно, что на первом, втором, третьем и четвертом местах находятся Oracle 21c, MySQL 8.0, Microsoft SQL Server 2019 и PostgreSQL 14.0 соответственно.

Сравним эти СУБД.

Сравнение начнем с Oracle.

Oracle Database или Oracle RDBMS — объектно-реляционная система управления базами данных компании Oracle.

Плюсы:

* Самые свежие инновации и впечатляющий функционал уже внедрены в этом продукте, поскольку компания Oracle стремится держать планку даже на фоне других разработчиков СУБД;
* СУБД от Oracle является крайне надёжной, фактически это эталон надёжности среди подобных систем;
* Oracle поддерживает самые большие базы данных. Большое количество пользователей для этой системы также не помеха. СУБД способна поддерживать любых пользователей, в любом количестве, которые при этом одновременно выполняют разные задачи. В Oracle не происходит соперничества между разными видами данных;
* Эта система обладает локальной управляемостью. Например, чтобы перезагрузить данные какого-то определенного приложения, не обязательно отключать всю систему. Администратору достаточно выключить доступ к нужному приложению и выполнить с ним требуемые манипуляции;
* Данная СУБД легко переносится с одной ОС на другую. Приложения, которые были разработаны специально для Oracle, легко переносятся на любую операционную систему с минимальными изменениями, а иногда даже без них.

Минусы:

* Стоимость Oracle может оказаться непомерно высокой, особенно для небольших организаций;
* Система может потребовать значительных ресурсов уже сразу после установки, поэтому возможно потребуется модернизировать оборудование для внедрения Oracle;
* Oracle не так просто настроить. Компании, которые платят большие деньги, обычно держат в штате штатных администраторов Oracle.

Следующим рассмотрим MySQL.

MySQL 8.0 - свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

Плюсы:

* Распространяется бесплатно;
* Прекрасно документирована;
* Предлагает много функций, даже в бесплатной версии;
* Пакет MySQL включен в стандартные репозитории наиболее распространённых дистрибутивов операционной системы Linux, что позволяет устанавливать её элементарно;
* Поддерживает набор пользовательских интерфейсов;
* Может работать с другими базами данных, включая DB2 и Oracle;
* Безопасность - большое количество функций, обеспечивающих безопасность, которые поддерживается по умолчанию;
* Масштабируемость - MySQL легко работает с большими объемами данных и легко масштабируется;
* Скорость - упрощение некоторых стандартов позволяет MySQL значительно увеличить производительность.

Минусы:

* Придётся потратить много времени и усилий, чтобы заставить MySQL выполнять несложные задачи, хотя другие системы делают это автоматически, например, создавать инкрементные резервные копии;
* Отсутствует встроенная поддержка XML или OLAP;
* Для бесплатной версии доступна только платная поддержка;
* Медленная разработка - Хотя MySQL технически открытое ПО, существуют жалобы на процесс разработки. Стоит заметить, что существуют другие довольно успешные СУБД, созданные на базе MySQL, например, MariaDB.

Теперь рассмотри Microsoft SQL Server.

Microsoft SQL Server 2019 - система управления реляционными базами данных (РСУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов - Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия.

Плюсы:

* Продукт очень прост в использовании;
* Текущая версия работает быстро и стабильно;
* Движок предоставляет возможность регулировать и отслеживать уровни производительности, которые помогают снизить использование ресурсов;
* Вы сможете получить доступ к визуализации на мобильных устройствах;
* Он очень хорошо взаимодействует с другими продуктами Microsoft;
* В SQL Server есть репликации через интернет, предусмотрена синхронизация. Есть полноценный веб-ассистент для форматирования страниц;
* В систему интегрирован сервер интерактивного анализа для принятия решений, создания корпоративных отчетов. Есть службы преобразования информации.

Минусы:

* Цена для юридических лиц оказывается неприемлемой для большей части организаций;
* Даже при тщательной настройке производительности корпорация SQL Server способен занять все доступные ресурсы;
* Сообщается о проблемах с использованием службы интеграции для импорта файлов.

Последней рассмотрим PostgreSQL.

PostgreSQL 14.0- свободная объектно-реляционная система управления базами данных.

Плюсы:

* Является масштабируемым и способен обрабатывать терабайты данных;
* Поддерживает формат json;
* Существует множество предопределенных функций;
* Доступен ряд интерфейсов.

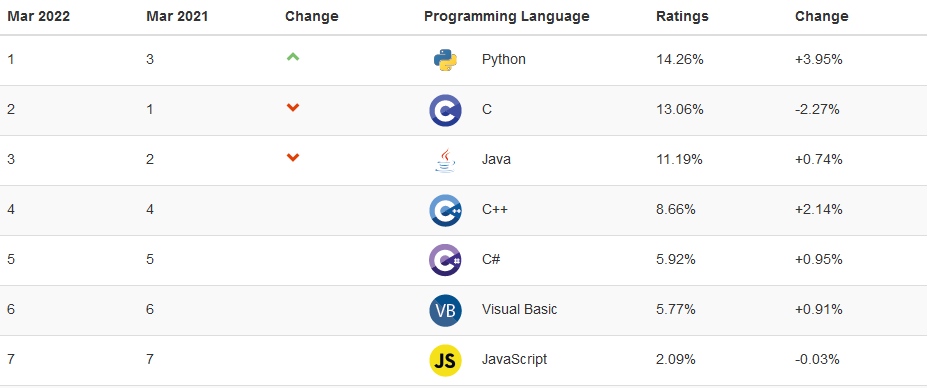
Минусы:

* Документация туманна, поэтому, возможно, ответы на некоторые вопросы придется искать в интернете;
* Конфигурация может смутить неподготовленного пользователя;
* Скорость работы может падать во время проведения пакетных операций или выполнения запросов чтения.

В итоге для разработки была выбрана Microsoft SQL Server 2019 удобности использования, масштабируемости и возможности взаимодействовать с другими продуктами Microsoft.

Дальше приступим к выбору языка программирования для проектирования.

Выбор начнем с рейтинга языков программирования с сайта tiobe.com[[3]](#footnote-3).



**Рисунок 2.3** – график популярности языков программирования

Сравним языки программирования Java SE 17, Python 3.10.2, C# 10.0.

Первым рассмотрим Java SE 17**.**

С момента своего появления в 1995 году считается одним из лучших языков для разработки. Язык программирования остается популярным среди разработчиков в течение многих лет.

Java, с помощью Node.js, может использоваться в различных приложениях, построенных на разных платформах. Более того, разработчики используют этот язык программирования с HTML, CSS и AJAX для разработки высококачественных приложений.

Плюсы:

* В настоящее время Java используется практически повсеместно в Интернете;
* Может использоваться параллельно с другими языками программирования для улучшения качества приложения;
* Обеспечивает большую функциональность в пользовательских интерфейсах;
* Повышенная производительность благодаря встроенному механизму обмена программными данными между несколькими компьютерами;
* Возможность автоматического управления памятью с одновременным и независимым запуском потоков;
* Обратная совместимость.

Минусы:

* Более низкая скорость, если сравнивать с C и C++. Как и у других языков высокого уровня, низкая производительность Java связана с компиляцией и абстракцией с помощью виртуальной машины. Среди других причин снижения скорости — функция очистки памяти, которая ведёт к потере производительности, когда требует больше 20 % загрузки процессора;
* Платные обновления для бизнеса и коммерческого использования. Стоимость зависит от количества пользователей или компьютеров;
* Многословный и сложный код.

Следующим рассмотрим Python — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Необычной особенностью языка является выделение блоков кода пробельными отступами.

Плюсы:

* Простой в написании код;
* Большое количество встроенных и внешних библиотек и фреймворков;
* Открытый исходный код;
* Большое сообщество программистов.

Минусы:

* Низкая скорость работы по сравнению с аналогами;
* Более высокое потребление памяти написанных программ по сравнению с аналогами;
* Скрипты компилируются каждый раз во время выполнения кода;
* Область видимости языка динамически ограничена.

Последним рассмотрим C# 10.0.

C# 10.0 - объектно-ориентированный язык программирования. Разработан в 1998 - 2001 годах группой инженеров компании Microsoft под руководством Андерса Хейлсберга и Скотта Вильтаумота как язык разработки приложений для платформы Microsoft .NET Framework. Впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

Плюсы:

* Кроссплатформенность;
* Множество готовых конструкций для использования программистом.

Минусы:

* Довольно большая стоимость. Бесплатность только для небольших компаний, учащихся и программистов-одиночек;
* C# менее гибкий, чем C++. C# сильно зависит от .NET framework;
* Все переменные (за исключением нескольких встроенных типов значений) являются ссылками, и освобождение памяти неявно выполняется с помощью сборщика мусора. Это затрудняет рассуждения об использовании памяти и общей производительности больших сложных приложений.

Выбор языка для разработки выпал C# 10.0. По причине множества готовых конструкций для использования, что в свою очередь упростит разработку приложения.

После выбора языка программирования, приступим к выбору среды разработки.

При выборе среды разработки выбор был между Project Rider 2021.2.2, Visual Studio 17.0.5 и Eclipse 4.22.0.

Первым рассмотрим Project Rider 2021.2.2

Project Rider или JetBrains Rider — кроссплатформенная интегрированная среда разработки программного обеспечения для платформы .NET, разрабатываемая компанией JetBrains. Поддерживаются языки программирования C#, VB.NET и F#.

Плюсы:

* ReSharper. Это плагин, изначально разработанный для повышения производительности Visual Studio. Теперь на его основе выпущена IDE.
* Поддержка полного цикла. Фирменная черта продуктов JetBrains, воплощенная и в Project Rider.
* Функциональность. Project Rider позволяет подключить MSBuild и XBuild, работать с CLI-проектами и организовать отладку приложений .NET and Mono. Множество опций для быстрого создания кода улучшает производительность.
* Кроссплатформенность. Project Rider работает с Windows, Linux и MacOS.
* Контроль версий. Встроенный инструмент позволяет напрямую организовать работу с Git, Mercurial и TFS.

Минусы:

* Молодость. Часть функциональности еще в разработке, не все стартовые баги исправлены.
* Высокая стоимость.

Следующим рассмотрим Visual Studio 17.0.5.

Visual Studio 17.0.5 - линейка продуктов компании Microsoft, включающих интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры, и приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Плюсы:

* Функциональная. В Visual Studio множество качественных плагинов. С их помощью можно расширить функциональность приложения и подключить другие языки;
* Официальная среда разработки для С #;
* Поддерживает платформы .NET. Visual Studio имеет широкие возможности по разработке приложений под Windows, в том числе в .NET-сегменте;
* Облачные хранилища. Зарегистрируйтесь в сообществе Visual Studio — и получите доступ к облачному хранилищу, где сможете располагать файлы проектов.

Минусы:

* Сложность. Самостоятельно освоить Visual Studio новичку будет непросто — слишком много доступных функций, спрятанных в подразделах меню;

Следующим рассмотрим Eclipse 4.22.0.

Eclipse 4.22.0 — свободная интегрированная среда разработки модульных кроссплатформенных приложений. Развивается и поддерживается Eclipse Foundation.

Наиболее известные приложения на основе Eclipse Platform — различные «Eclipse IDE» для разработки ПО на множестве языков.

Плюсы:

* Множество плагинов.
* Активное сообщество;
* Отличные компилятор и отладчик. Первый работает на порядок быстрее, чем у конкурентов, второй — показывает потоки, пересечения, позволяет гибко управлять ходом отладки;
* Кастомизация. Благодаря плагинам и настройкам можно полностью персонализировать Eclipse;
* Бесплатность;
* Высокая функциональность. Благодаря разработчикам и членам сообщества с помощью Eclipse можно провести любой C# - продукт по полному циклу разработки.

Минусы:

* Сложность. Как и любой функциональный продукт, Eclipse может показаться новичку слишком сложным;
* Нет гарантий надежности. Так как плагины создаются сообществом, за их качество отвечает только разработчик. Кроме того, сами создатели Eclipse с каждой новой версией плодят баги, не успевая порой исправлять старые.

В итоге рассмотрев все плюсы и минусы было принято решение использовать Visual Studio 17.0.5. По причине высокой совместимости с языком С #.

Теперь необходимо определиться с технологией для разработки приложения.

Сравним технологии для разработки WPF и WinForms.

Первым рассмотрим WPF.

Windows Presentation Foundation (WPF) - это бесплатная графическая подсистема с открытым исходным кодом, первоначально разработанная Microsoft для визуализации пользовательских интерфейсов в приложениях на базе Windows. WPF, ранее известный как "Авалон", был первоначально выпущен как часть .NET Framework 3.0 в 2006 году.

В WPF элементы пользовательского интерфейса разработаны в XAML, а поведение может быть реализовано на процедурных языках, таких как C # и VB.Net. Так что очень легко отделить поведение от кода дизайнера.

Далее рассмотрим WinForms.

WinForms — интерфейс программирования приложений, отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки.

В состав Windows Forms входят многофункциональные элементы пользовательского интерфейса, позволяющие воссоздавать возможности таких сложных приложений, как Microsoft Office. Используя элементы управления ToolStrip и MenuStrip, можно создавать панели инструментов и меню, содержащие текст и рисунки, подменю и другие элементы управления, такие как текстовые поля и поля со списками.

В итоге анализа была выбрана WPF, по причине более широкой настройки интерфейсов и удобства использования, что позволит сделать интерфейс более гибким.

После выбора средств реализации для приложения, приступим к выбору средств реализации для разработки сайта.

Выбор начнем с языков разметки.

При выборе языка разметки выбор был между HTML, XML, Wiki.

Первым рассмотрим HTML.

Больше всего распространён язык HTML (аббревиатура от английского Hypertext Markup Language − «язык разметки гипертекста»). Он является стандартным для интернет-документов, с его помощью создаются все веб-страницы. Документы, которые содержат в себе специальный код, обрабатываются браузерами и представляются пользователям в удобном интерфейсе. Таким образом, вы получаете возможность просматривать страницы, распечатывать их, использовать для передачи данных на серверы.

Следующим рассмотрим XML.

XML (в переводе с английского eXtensible Markup Language − расширяемый язык разметки). Название связано с тем, что он не имеет зафиксированного формата. Это значит, что пользователи могут создавать собственные теги, которые позволяют глубоко обработать текстовый документ. Как правило, XML используется с целью осуществления описания грамматики других языков, а также обеспечения контроля над правильностью составления документа. Внешне документы HTML и XML очень похожи. Принципиальное отличие − более высокие требования к тегам во втором случае (например, при простановке тега нужно следить за регистром).

Следующим рассмотрим Wiki.

Wiki-разметка используется для тех веб-страниц, которые могут правиться (дополняться, редактироваться) пользователями. Это позволяет создавать тексты даже тем, кто не разбирается в особенностях HTML-разметки. Безусловно, в этом языке существуют свои правила, но они предельно просты, все изменения можно просмотреть и при необходимости вернуться к первоначальной версии. Благодаря этому исправить ошибки гораздо проще. Администраторы могут ограничить права редактирования размещённых текстов, разрешить проведение манипуляций только определённым пользователям и т. д. Отличительная особенность этой разметки − гипертекстовость (связь документов посредством проставления контекстных гиперссылок).

После рассмотрения языков разметки выбор сделан в пользу HTML из-за распространенности и простоты использования.

После выбора средств реализации приступим к разделу проектирования.

2.4 Технические требования к разрабатываемому комплексу программного обеспечения

Результатами разработки комплекса ПО автострахования будут является: исполняемый файл, руководство пользователя, папки с программными данными, разработанная база данных и сайт.

Для корректной работы программного обеспечения не обходимо иметь ПК, соответствующий следующим требованиям:

* операционная система семейства Windows, версия не ниже Windows 10;
* пакет Microsoft Office старше 2010 года;
* периферийные устройства (мышь и клавиатура);
* монитор;
* персональный компьютер.

В таблице 1 представлены минимальные аппаратные требования для OC Windows 10.

**Таблица 1**

**Минимальные системные требования:**

|  |  |
| --- | --- |
| Операционная система | Windows 10 |
| Процессор | 2.4 ГГц |
| Оперативная память | 2 ГБ |
| Драйвера видеокарты | DirectX 9 |
| Место на диске | 10 ГБ |

2.4.1 Проектирование диаграммы прецедентов, концепт модели

Диаграмма вариантов использования в UML — диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Основное назначение диаграммы — описание функциональности и поведения, позволяющее заказчику, конечному пользователю и разработчику совместно обсуждать проектируемую или существующую систему.

Use – case диаграмма изображена в Приложении 2 Рисунок 1.

Далее спроектируем концептуальную модель.

Концептуальная модель - это модель, представленная множеством понятий и связей между ними, определяющих смысловую структуру рассматриваемой предметной области или её конкретного объекта.

Основная задача концептуальной модели - передать фундаментальные принципы и основные функциональные возможности системы, которую она представляет. Кроме того, концептуальная модель должна быть разработана таким образом, чтобы обеспечить легко понятную системную интерпретацию для пользователей модели. Концептуальная модель, если она реализуется должным образом, должна удовлетворять четырем основным целям.

Концептуальная модель изображена в ПРИЛОЖЕНИИ 2 Рисунок 2.

Далее приступим к проектированию инфологической модели.

2.4.2 Проектирование инфологической модели и словаря данных

Инфологическая модель (информационно-логическая модель) - ориентированная на человека и не зависимая от типа СУБД модель предметной области, определяющая совокупности информационных объектов, их атрибутов и отношений между объектами, динамику изменений предметной области, а также характер информационных потребностей пользователей. Инфологическая модель предметной области может быть описана моделью "сущность—связь", в основе которой лежит деление реального мира на отдельные различимые сущности, находящиеся в определенных связях друг с другом, причем обе категории - сущность и связь полагаются первичными, неопределенными понятиями.

Инфологическая модель изображена в ПРИЛОЖЕНИИ 2 Рисунок 3.

Словарь данных представлен ниже (смотреть Табл. 2 – 19).

**Таблица 2 - ОСАГО**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Osago** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | Id\_osago | Int | y | AUTO INCREMENT |
| FK | Id\_driver | int | y | Foreign key to Osago from Driver |
| FK | Id\_car | Int | y | Foreign key to osago from Car |
| FK | Id\_worker | int | y | Foreign key to osago from Worker |
| FK | Id\_status\_osago | Int | Y | Foreign key to Osago from Status\_osago |
|  | Base\_rate | Decimal(9,2) |  |  |
|  | coefficient\_territory\_of\_use | Decimal(9,2) |  |  |
| FK | id\_coefficient\_inserance\_compensation | Int |  | Foreign key to Osago from Coefficient\_insurance\_compensation |
| FK | id\_coefficient\_engine\_power | Int |  | Foreign key to Osago from Coefficient\_engine\_power |
| FK | id\_coefficient\_restriction\_use | Int | Y | Foreign key to Osago from Coefficient\_restriction\_use |
| FK | id\_coefficient\_experience\_age | Int |  | Foreign key to Osago from Coefficient\_experience\_age |
| FK | id\_coefficient\_period\_of\_use | Int | Y | Foreign key to Osago from Coefficient\_period\_of\_use |
| FK | id\_coefficient\_insurance\_period | Int | Y | Foreign key to Osago from Insurance\_period |
|  | price | Decimal(9,2) |  |  |

**Таблица 3 – Статус ОСАГО**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Status\_osago** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | id\_status\_osago | int | y | AUTO INCREMENT |
|  | status\_osago | Nchar(30) | y |  |

**Таблица 4 – Коэффициент страховой компенсации**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Coefficient\_insurance\_compensation** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | Id\_coefficient\_insurance\_compensation | int | y | AUTO INCREMENT |
|  | coefficient\_insurance\_compensation | Decimal(9,2) | Y |  |
|  | number\_insurance\_compensation | int | Y |  |
|  | coefficient\_insurance\_compensation | Decimal(9,2) | y |  |

**Таблица 5 – Коэффициент мощности двигателя**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Coefficient\_engine\_power** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | id\_coefficient\_engine\_power | int | y | AUTO INCREMENT |
|  | engine\_power | nchar(30) | y |  |
|  | coefficient\_engine\_power | Decimal(9,2) | y |  |
|  | engine\_power\_number | int | y |  |

**Таблица 6 – Коэффициент ограничения количества лиц**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Coefficient\_restriction\_use** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | id\_coefficient\_restriction\_use | int | y | AUTO INCREMENT |
|  | restriction\_use | Nchar(10) | y |  |
|  | coefficient\_restriction\_use\_physical | Decimal(9,2) | y |  |
|  | coefficient\_restriction\_use\_juridical | Decimal(9,2) | y |  |

**Таблица 7 – Коэффициент возраста и стажа вождения**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Coefficient\_experience\_age** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | id\_coefficient\_experience\_age | Int | Y | AUTO INCREMENT |
|  | age | nchar(10) | Y |  |
|  | experience | nchar(10) | Y |  |
|  | coefficient\_experience\_age | Decimal(9,2) | y |  |

**Таблица 8 – Коэффициент периода использования транспорта**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Coefficient\_period\_of\_use** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | id\_coefficient\_period\_of\_use | int | y | AUTO INCREMENT |
|  | period\_of\_use | Nchar(20) | y |  |
|  | id\_coefficient\_period\_of\_use | Decimal(9,2) | y |  |

**Таблица 9 – Коэффициент периода страховки**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Insurance\_period** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | id\_coefficient\_insurance\_period | int | y | AUTO INCREMENT |
|  | insurance\_period | Nchar(20) | y |  |
|  | coefficient\_insurance\_period | Decimal(9,2) | y |  |

**Таблица 10 – Водители**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Driver** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | id\_driver | int | y | AUTO INCREMENT |
|  | name | Nchar(60) | y |  |
|  | surname | Nchar(60) |  |  |
|  | patronymic | Nchar(60) |  |  |
|  | date\_birth | date | y |  |
|  | driving\_experience | int | y |  |
|  | number\_insurance\_compensation | int | y |  |
|  | coefficient\_beginning | decimal(9, 2) | y |  |
|  | passport\_series | int | y |  |
|  | passport\_number | int | y |  |
| FK | id\_face | int | y | Foreign key to Driver from Face |
| FK | id\_gender | int | y | Foreign key to Driver from Gender |

**Таблица 11 – Лицо**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Face** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | Id\_face | int | y | AUTO INCREMENT |
|  | face | Nchar(40) | y |  |

**Таблица 12 – Пол**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Gender** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | Id\_gender | int | y | AUTO INCREMENT |
|  | gender | Nchar(1) | y |  |

**Таблица 13 – Работник**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Worker** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | Id\_worker | int | y | AUTO INCREMENT |
|  | Name | Nchar(60) | y |  |
|  | Surname | Nchar(60) | y |  |
|  | Patronymic | Nchar(60) |  |  |
|  | INN | Nchar(12) | y |  |
|  | education | Nchar(60) |  |  |
|  | login | Nchar(30) | y |  |
|  | password | Nchar(30) | y |  |
|  | Work\_experience | Nchar(20) |  |  |
|  | Telephone\_number | Nchar(20) |  |  |
| FK | Id\_gender | Int | y | Foreign key to Worker from Gender |
| FK | Id\_role | Int | y | Foreign key to Worker from Role |

**Таблица 14 – Роль**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Role** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **Notes** |
| PK | Id\_role | Int | y | AUTO INCREMENT |
|  | role | Nchar(40) | y |  |

**Таблица 15 – Машина**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Car** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | Id\_car | int | y | AUTO INCREMENT |
|  | Car\_number | Nchar(6) | y |  |
|  | id\_region\_number | int |  | Foreign key to Car from Region\_number |
|  | Engine\_power | nchar(10) | y |  |
|  | Vin\_car\_number | Nchar(17) | y |  |
|  | Release\_year | date | y |  |

**Продолжение таблицы 15**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FK | Id\_color | int | y | Foreign key to Car from Color |
| FK | Id\_model | int | y | Foreign key to Car from Model |
| FK | Id\_car\_category | int | y | Foreign key to Car from Car\_category |

**Таблица 16 – Цвет**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Color** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | Id\_color | int | y | AUTO INCREMENT |
|  | color | Nchar(30) | y |  |

**Таблица 17 – Номер региона**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Region\_number** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | id\_region\_number | int | y | AUTO INCREMENT |
|  | number\_region | int | y |  |
|  | region | nchar(60) | y |  |
|  | coefficient\_territory\_of\_use | decimal(9, 2) | y |  |

**Таблица 18 – Модель Автомобиля**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
| PK | Id\_model | Int | y |  |
|  | model | Nchar(40) | y |  |

**Таблица 19 – Категория автомобиля**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Car\_category** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
|  | Id\_car\_category | Int | y |  |
|  | Car\_category | Nchar(10) | y |  |
|  | Base\_rate | Decimal(9,2) | y |  |

**Таблица 20 – Марка автомобиля**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Car\_brand** | | | | |
| **Key** | **fieldName** | **dataType** | **required** | **notes** |
|  | Id\_car\_brand | Int | y |  |
|  | Car\_brand | Nchar(10) | y |  |
|  | Id\_model | Int | y |  |

2.4.3 Физическая модель

Физическая модель данных (ФМД) – это модель данных, описанная с помощью средств конкретной СУБД. ФМД строится на базе инфологической путем добавления особенностей конкретной СУБД. К таким особенностям могут относиться поддерживаемые СУБД типы данных, соглашения о присвоении имен таблицам, атрибутам и т.д. ФМД фактически является готовым заданием на создание БД, имея которое можно реализовать БД в выбранной СУБД.

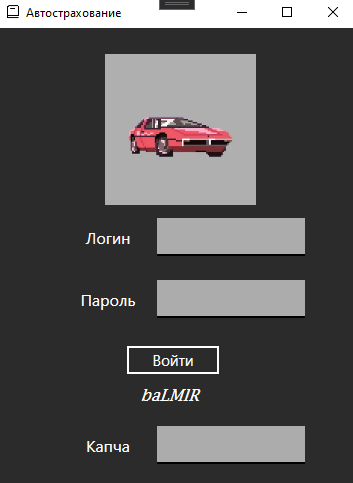
Физическую модель БД изображена в ПРИЛОЖЕНИИ 2 рисунок 3.

2.5 Разработка интерфейса

2.5.1 Разработка интерфейса ПО

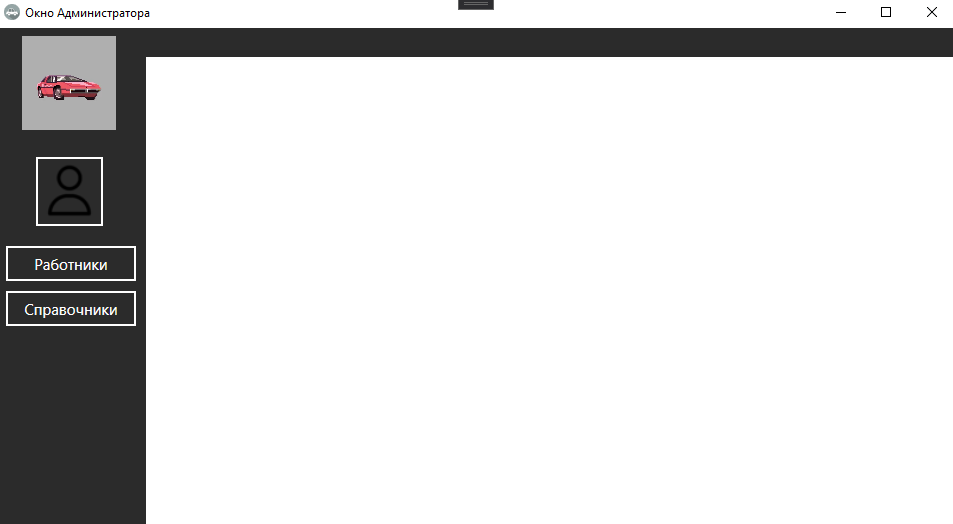
При проектировании интерфейса были использованы поля ввода текста, выпадающие списки, а также функциональные кнопки. Некоторые из них генерируются с помощью кода.

На рисунке 2.4 изображена страница авторизации. На форме использовано 3 Label, 2 TextBox, 1 PasswordBox, 1 Button, 1 Image. При правильном вводе логина и пароля и нажатии на кнопку происходит открытие окна в зависимости от роли авторизовавшегося.



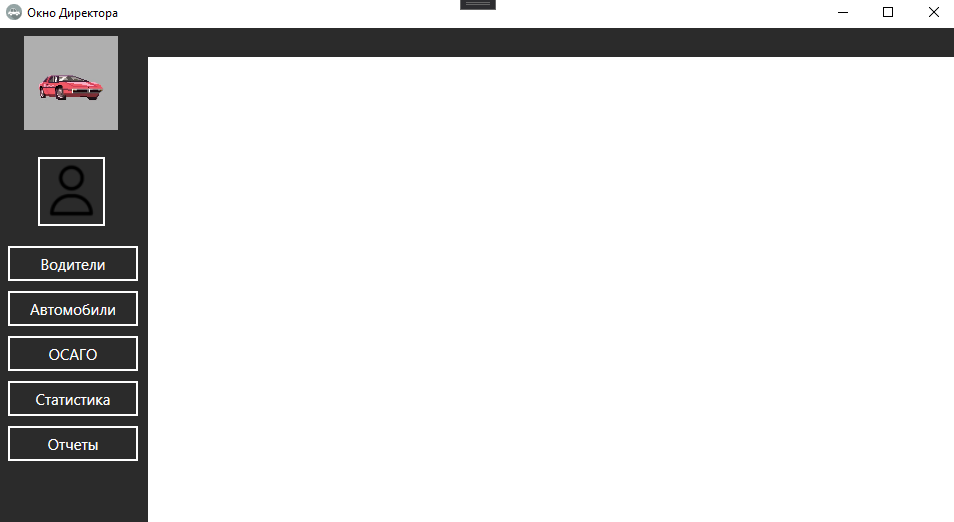
**Рисунок 2.4** – форма авторизации

На странице 2.5 изображена форма окна администратора. На ней присутствуют кнопки для навигации между фреймами. При нажатии на кнопки происходит выбор фрейма. С данной формой может работать сотрудник с ролью администратор.



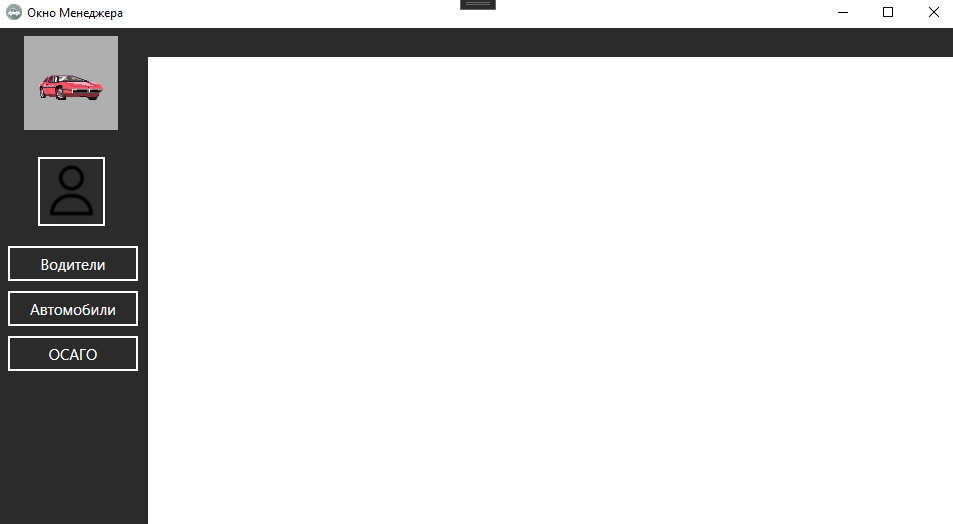
**Рисунок 2.5** – форма окна администратора

На рисунке 2.6 изображена форма окна директора. На ней присутствуют кнопки для навигации между фреймами. При нажатии на кнопки происходит выбор фрейма. С данной формой может работать сотрудник с ролью директора.



**Рисунок 2.6** – форма окна директора

На рисунке 2.7 изображена форма окно менеджера. На ней присутствуют кнопки для навигации между фреймами. При нажатии на кнопки происходит выбор фрейма. С данной формой может работать сотрудник с ролью менеджера.



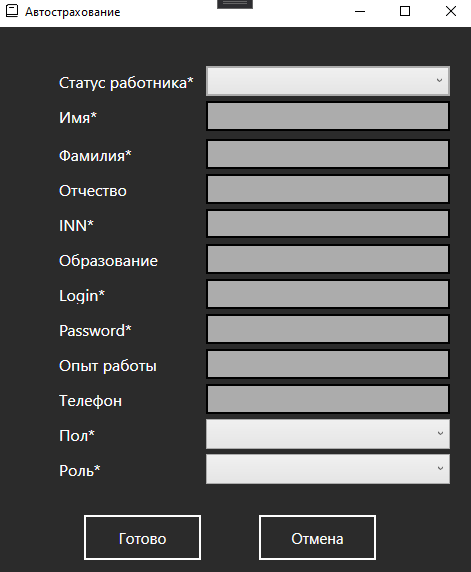
**Рисунок 2.7** – форма окна менеджера

На рисунке 2.8 изображена форма профиля. На ней присутствуют текстовые блоки в которых отображается информация о работнике.



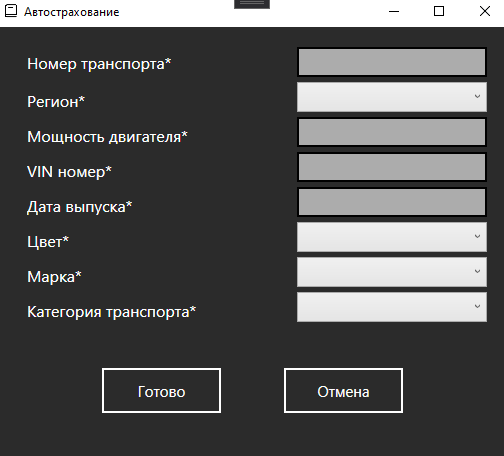
**Рисунок 2.8** – форма профиля

На рисунке 2.9 изображена форма добавления работников фирмы. На форме присутствуют 3 ComboBox, 9 TextBox, 12 Label, 2 Button. При нажатии на кнопку происходит добавление работника.



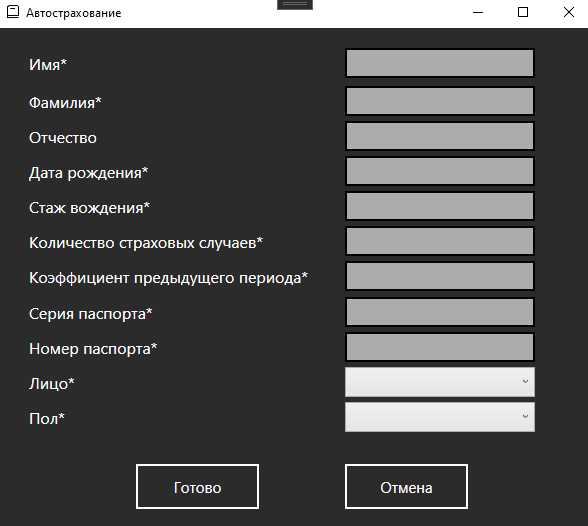
**Рисунок 2.9** – форма добавления работника

На рисунке 2.10 изображена форма добавления транспорта. На форме присутствуют 4 ComboBox, 4 TextBox, 8 Label, 2 Button. При нажатии на кнопку происходит добавление транспорта.



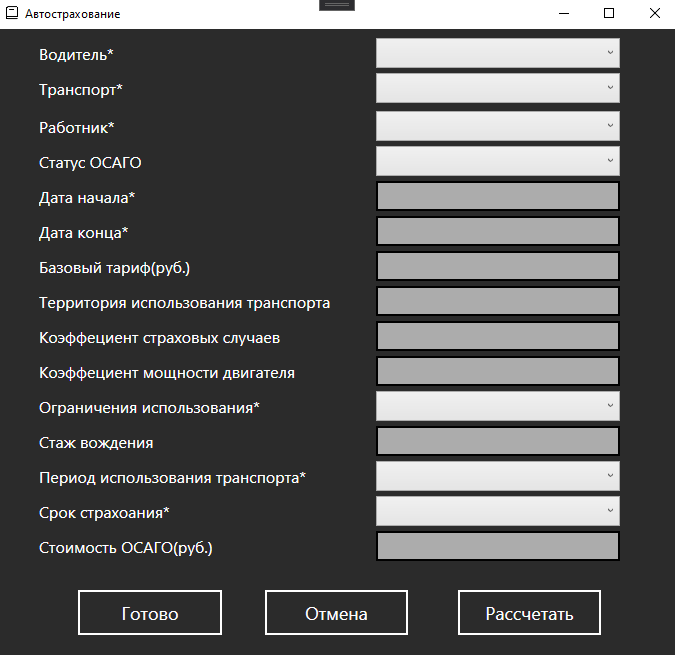
**Рисунок 2.10** – форма добавления транспорта

На рисунке 2.11 изображена форма добавления клиентов. На форме присутствуют 2 ComboBox, 9 TextBox, 11 Label, 2 Button. При нажатии на кнопку происходит добавление клиента.



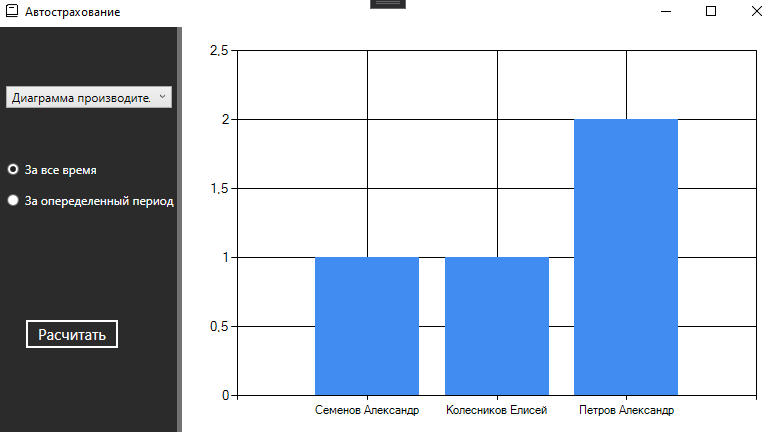
**Рисунок 2.11** – форма добавления клиентов

На рисунке 2.12 изображена форма добавления ОСАГО. На форме присутствуют 7 ComboBox, 8 TextBox, 16 Label, 3 Button. При нажатии на кнопку происходит добавление ОСАГО.



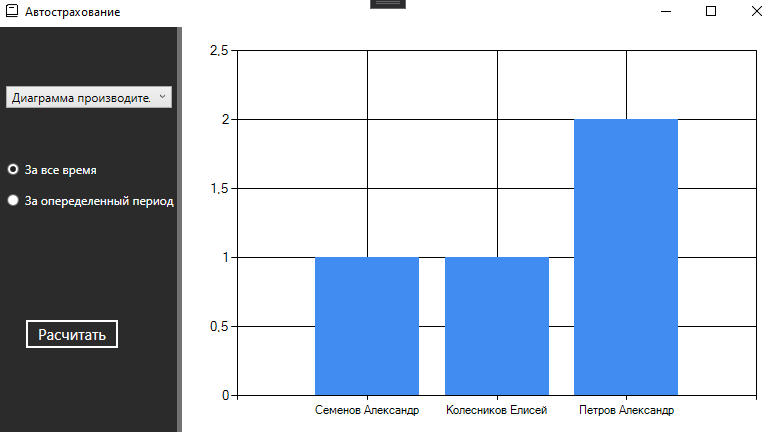
**Рисунок 2.12** – форма добавления ОСАГО

На рисунке 2.13 изображена форма статистики. На форме присутствуют 1 ComboBox, 2 RadioButton, 2 DatePicker , 2 Label, 1 Button.



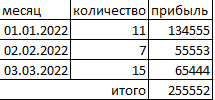
**Рисунок 2.13** – форма статистики

На рисунке 2.14 изображена форма составления отчета. На форме имеются кнопки для составления отчета.



**Рисунок 2.14** – форма составления отчета

На рисунке 2.15 изображен отчет, составленный директором.

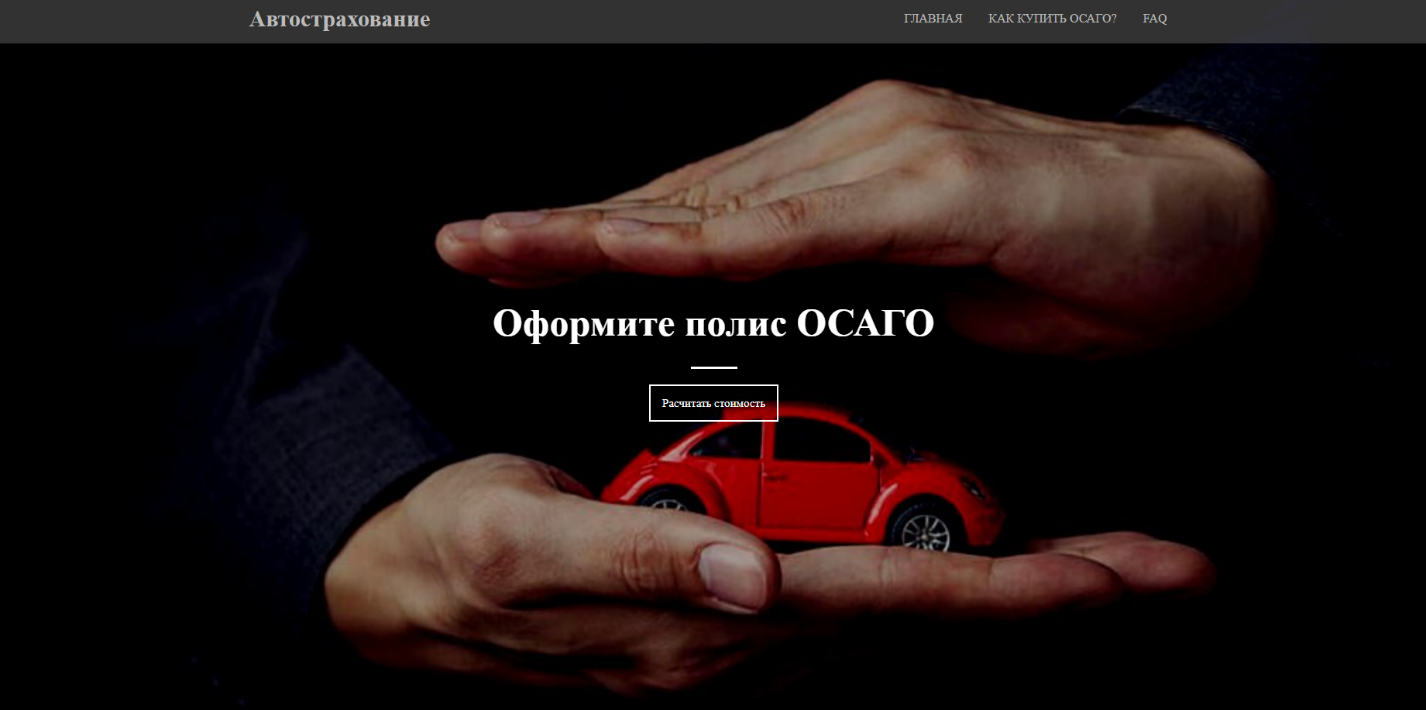


**Рисунок 2.15** – отчет

В следующем разделе будет рассмотрен интерфейс сайта.

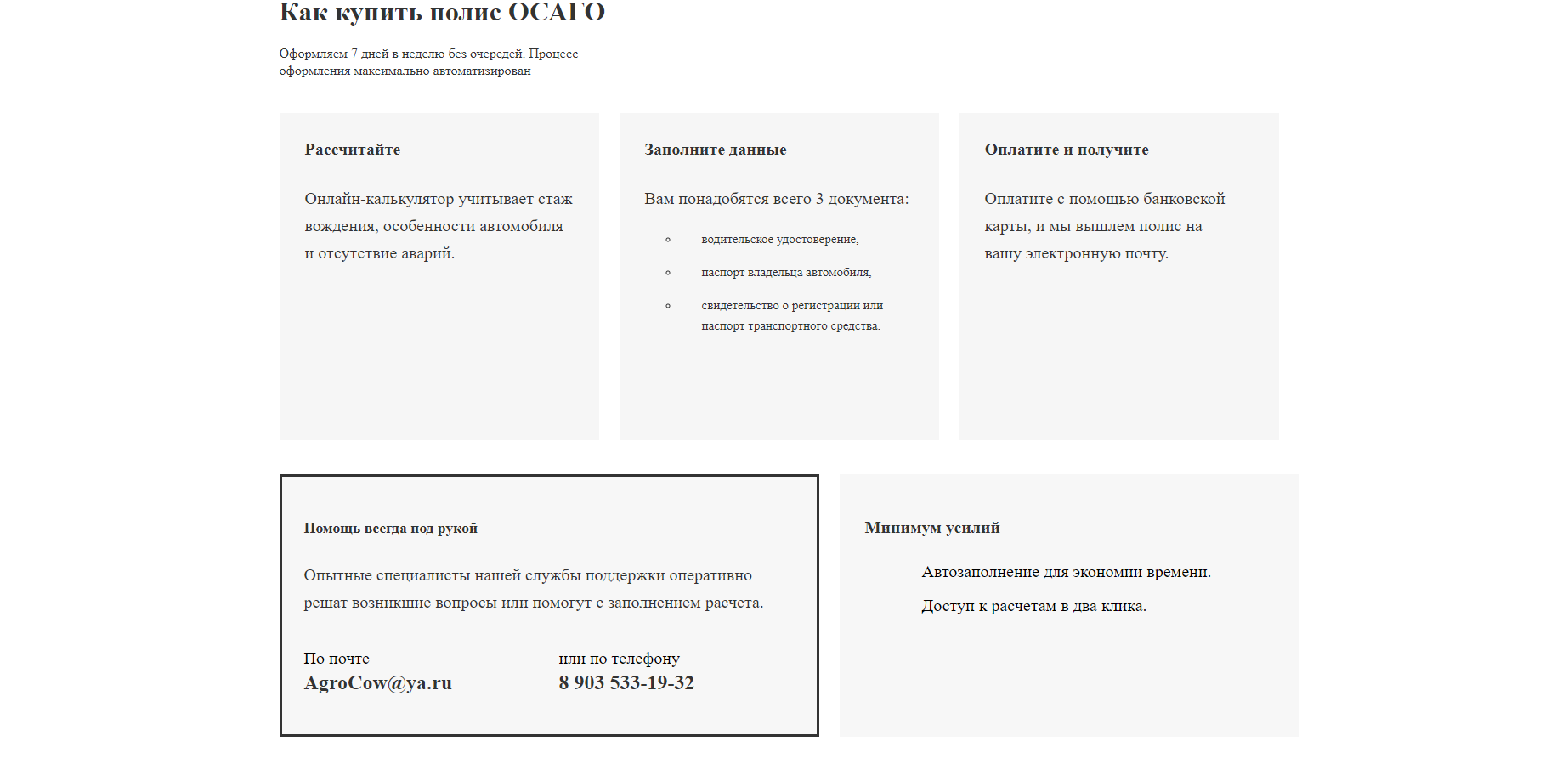
2.5.2 Разработка интерфейса сайта

При входе на сайт пользователя встречает шапка сайта, в которой имеется навигация и название компании. В дальнейшем эта шапка будет использована на каждой странице сайта.

****

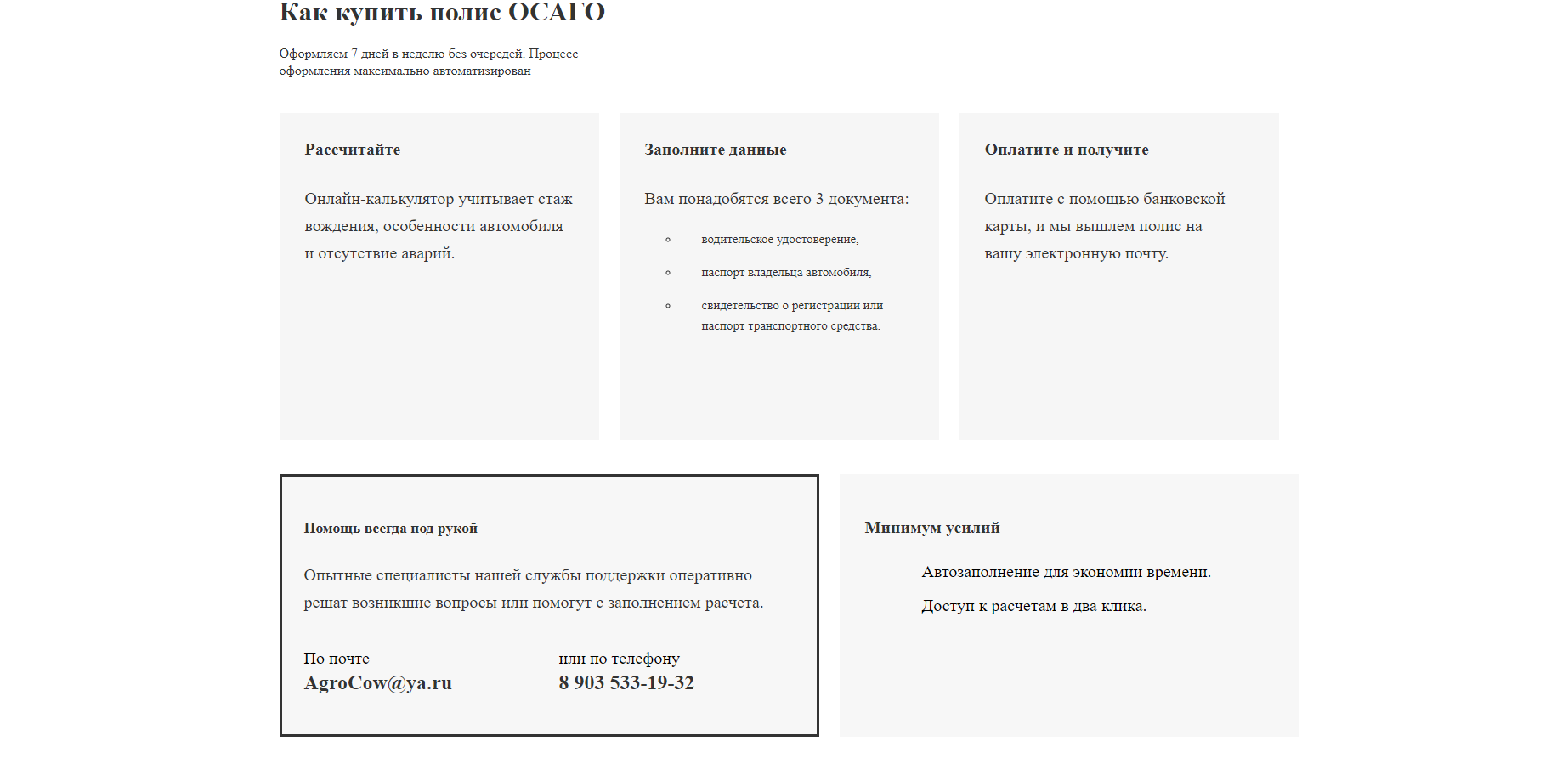
**Рисунок 2.16** – Верхняя часть сайта

Под шапкой начинается блок с помощью для посетителей сайта. Тут расписана информации о обратной связи, способе получения ОСАГО и некоторых преимуществах сайта.

****

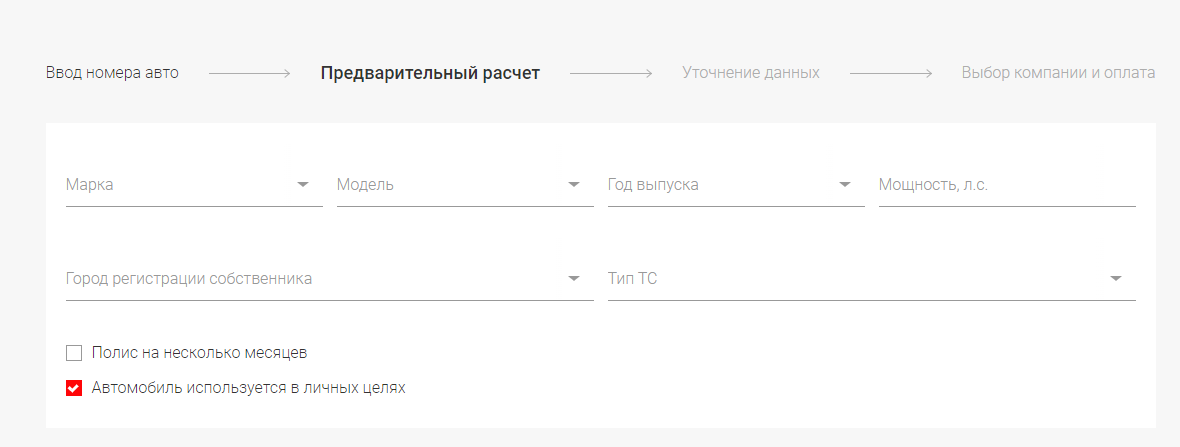
**Рисунок 2.17** – Блок помощи сайта

Под блоком помощи начинается блок с часто задаваемыми вопросами. Тут находятся ответы на большинство вопросов по ОСАГО.

****

**Рисунок 2.18** – Блок с часто задаваемыми вопросами

При нажатии на кнопку расчёта стоимости ОСАГО в верхней части сайта, клиента перебрасывает на страницу сайта с расчетом стоимости, показанную на рисунке 2.15, где необходимо заполнить данные.



**Рисунок 2.19** – страница сайта с расчетом стоимости

При заполнении всех данных происходит выбор оплаты и после оплаты на почту, указанную в данных присылается готовый полис ОСАГО. Клиент может распечатать полис и полноценно его использовать.

В следующем разделе приложение будет протестировано.

# **3. ЭКСПЕРЕМЕНТАЛЬНЫЙ РАЗДЕЛ**

3.1 Тестирование комплекса ПО

Тестирование программного обеспечения — процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определённым образом.

Тестирование в первую очередь делится на уровни. Выделяют 4 основных уровня тестирования:

* Компонентное/модульное тестирование;
* Интеграционное тестирование;
* Системное тестирование;
* Приемочное тестирование.

Первым рассмотрим метод тестирование белого ящика.

Сначала разберёмся что же такое тестирование белого ящика.

Тестирование белого ящика — тестирование, которое учитывает внутренние механизмы системы или компонента.

Обычно включает тестирование ветвей, маршрутов, операторов. При тестировании выбирают входы для выполнения разных частей кода и определяют ожидаемые результаты.

Традиционно тестирование белого ящика выполняется на уровне модулей, однако оно используется для тестирования интеграции систем и системного тестирования, тестирования внутри устройства и путей между устройствами. Этот метод тестирования не может выявить невыполненные части спецификации, отсутствие требований или создание не того приложения.

Плюсы:

* Очень большая точность;
* Просто автоматизировать;
* Позволяет определить, где в коде произошла ошибка;
* Для выполнения тестирования не нужен UI.

Минусы:

* Инженеры, выполняющие тестирование, должны знать программирование на достаточном уровне;
* Иногда бывает невозможно протестировать каждый участок кода;
* Фокус на тестировании кода может привести к ситуации, когда пропускаются баги, связанные с функциональностью. При этом код может работать правильно, но приложение не будет соответствовать требованиям.

Теперь рассмотрим тестирование черного ящика.

Тестирование чёрного ящика или поведенческое тестирование — стратегия тестирования функционального поведения объекта с точки зрения внешнего мира, при котором не используется знание о внутреннем устройстве тестируемого объекта. Иначе говоря, тестированием чёрного ящика занимаются тестировщики, не имеющие доступ к исходному коду приложения. Под стратегией понимаются систематические методы отбора и создания тестов для тестового набора. Стратегия поведенческого теста исходит из технических требований и их спецификаций.

Плюсы:

* QA проводит тестирование, пользуясь приложением так, как это делал бы конечный пользователь;
* Позволяет найти несоответствия между спецификацией и разработанным приложением;
* QA не должен глубоко погружаться в особенности разработки и архитектуры;
* В рамках тестирования создается документация, включающая в себя тест-кейсы;
* Тест-кейсы могут быть автоматизированы.

Минусы:

* Требования или спецификация могут быть неточными и/или непонятными, поэтому могут быть трудности с написанием тест-кейсов;
* Может быть сложно протестировать все части приложения.

Последним рассмотрим тестирование серого ящика.

Проверка серого ящика – это метод тестирования программного продукта или приложения с частичным знанием его внутреннего устройства. Для выполнения тестирования серого ящика нет необходимости в доступе тестировщика к исходному коду. Тесты пишутся на основе знания алгоритма, архитектуры, внутренних состояний или других высокоуровневых описаний поведения программы.

Плюсы:

* Тестирование серого ящика включает в себя плюсы тестирования «черного» и «белого». Другими словами, тестировщик смотрит на объект тестирования с позиции «черного» ящика, но при этом проводит анализ на основе тех данных, что он знает о системе.
* Тестировщик может проектировать и использовать более сложные сценарии тестирования.
* Тестировщик работает совместно с разработчиком, что позволяет на начальном этапе убрать избыточные тест-кейсы. Это сокращает время функционального и нефункционального тестирования и положительно влияет на общее качество продукта.
* Предоставляет разработчику достаточно времени для исправления дефектов.

Минусы:

* Возможность анализа кода и тестового покрытия ограничена, так как доступ к исходному коду отсутствует.
* Тесты могут быть избыточными в том случае, когда разработчик также проверяет свой код Unit-тестами.
* Нельзя протестировать все возможные потоки ввода и вывода, поскольку на это требуется слишком много времени

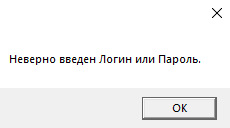
Рассмотрев все плюсы и минусы было выбрано тестирование белого ящика. Так как разрабатываемое приложение будет использовано только работниками компании нам менее важно тестировать интерфейс приложения чем внутреннюю работу кода.

В результате тестов проблемы обнаружены и устранены.

3.2 Отладка разработанного комплекса ПО

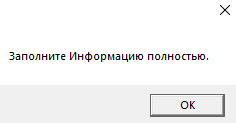
У всех форм используется защита от неправильного ввода в случае его некорректности он не будет отправлять запрос в БД, а предупредит что вы ввели неправильно или вообще ничего не ввели.

При неверном вводе логина или пароля при авторизации высвечивается сообщение, показанное на рисунке 3.1.



**Рисунок 3.1** – сообщение о неверном вводе логина или пароля

При неверном или неполном вводе данных при редактировании или добавлении данных высвечивается сообщение, показанное на рисунке 3.2.



**Рисунок 3.2** – сообщение о неверном вводе логина или пароля

# **ЗАКЛЮЧЕНИЕ**

В ходе разработки комплекса ПО учета автострахования была получена полностью рабочая программа и сайт.

В результате выполнения дипломного проекта был разработан комплекс ПО по автострахованию, также выполнены все поставленные задачи, а именно:

1. Cформулирована постановка задачи и исследовать предметную область;
2. Проанализированы существующие программные обеспечения по автоматизации деятельности компании по автострахованию;
3. Спроектирована концептуальная модель данных;
4. Спроектирована информационно-логическая модель данных;
5. Проанализировано и выбрано OC, СУБД и язык программирования;
6. Спроектирована физическая модель данных;
7. Разработан интерфейс комплекса программного обеспечения;
8. Определины технические и программные средства реализации проекта;
9. Проверена работоспособность программного обеспечения средствами тестирования;
10. Разработано руководство пользователя. ­

Реализация комплекса ПО была осуществлена при помощи технологии WPF с использованием языковых средств программирования С#. В качестве среды разработки была использована среда Microsoft Visual Studio 2022. Разработка базы данных выполнялась в СУБД Microsoft SQL Server 2019.

Таким образом цель курсового проекта была полностью достигнута и в соответствии с ней было полностью продемонстрированно рабочее программное обеспечение.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. ГОСТ 19.102-77 Единая система программной документации. Стадии разработки.
2. ГОСТ Р ИСО/МЭК ТО 15271-2002 Информационная технология. Процессы жизненного цикла программных средств
3. Борис Новиков, Екатерина Горшкова, Основы технологий баз данных, Litres - 2019, 240 с.
4. Дейт К.Дж. - Введение В Системы Баз Данных: Издательство: Диалектика, 2019 - 135с.
5. Кетов Д. В. К37 Внутреннее устройство Linux. — 2-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2021. — 17 с
6. Осипов Д. Л. Технологии проектирования баз данных. - М.: ДМК Пресс, 2019, 113 c.
7. Эдвард Сьорье, Проектирование и реализация систем управления базами данных, 2020 – 461 с.
8. Столлингс В. - Операционные системы. Внутренняя структура и принципы проектирования: 9-е изд. : Пер. с англ. - СПб. : ООО "Диалектика", 2020 - 83с.
9. Джон, Скит C#. Программирование для профессионалов / Скит Джон. - М.: Диалектика / Вильямс, **2018. - 586** c.
10. bestprogrammer [Электронный ресурс] – Режим доступа: https://bestprogrammer.ru/izuchenie/10-luchshih-java-ide-v-2021-godu
11. DB-engines [Электронный ресурс] - Режим доступа: https://db-engines.com/en/ranking
12. Habr [Электронный ресурс] - Режим доступа: https://habr.com/ru/company/qiwi/blog/346438/
13. Habr [Электронный ресурс] - Режим доступа: https://habr.com/ru/post/142385/
14. Intuit [Электронный ресурс] - Режим доступа: https://intuit.ru/studies/courses/3632/874/lecture/14297?page=5
15. StatCounter [Электронный ресурс] - Режим доступа: https://gs.statcounter.com/os-market-share/desktop/worldwide/#yearly-2021-2021-bar
16. StudFile [Электронный ресурс] - Режим доступа: https://studfile.net/preview/1444533/page:9/
17. Studopedia [Электронный ресурс] - Режим доступа: https://studopedia.su/15\_43262\_preimushchestva-inkrementnoy-modeli.html
18. Suhorukov [Электронный ресурс] - Режим доступа: http://suhorukov.com/news\_akademy/sravnenie-operacionnyh-sistem-mac-os-linux-i-windows
19. Quality Lab [Электронный ресурс] - Режим доступа: https://quality-lab.ru/blog/key-principles-of-gray-box-testing/
20. Testengineer [Электронный ресурс] - Режим доступа: https://testengineer.ru/testirovanie-belogo-chernogo-yashchika/

# **ПРИЛОЖЕНИЕ 1**

**Листинг запросов**

Функция disp\_data – выводит данные в DataGrid и заполняет коллекции.

public void disp\_data()

{

string query = "SELECT \* FROM Osago";

try

{

dataBase2.openConnection();

SqlDataAdapter SDA = new SqlDataAdapter(query, dataBase2.getConnection());

DataTable dt = new DataTable();

SDA.Fill(dt);

OsagoGrid.ItemsSource = dt.DefaultView;

count = dt.Rows.Count;

}

catch (Exception ex)

{

MessageBox.Show("Ошибка соединения с базой данных. \n\n\n\nПодрабнее:\n" + ex.ToString());

}

finally

{

dataBase2.closeConnection();

}

try

{

SqlCommand sqlCommand = new SqlCommand(query, dataBase2.getConnection());

dataBase2.openConnection();

SqlDataReader reader = sqlCommand.ExecuteReader();

while (reader.Read())

{

osago.Add(new DataClassOsago

{

id\_osago = Convert.ToInt32(reader[0].ToString()),

id\_driver = Convert.ToInt32(reader[1].ToString()),

id\_car = Convert.ToInt32(reader[2].ToString()),

id\_worker = Convert.ToInt32(reader[3].ToString()),

id\_status\_osago = Convert.ToInt32(reader[4].ToString()),

start\_date = reader[5].ToString(),

end\_date = reader[6].ToString(),

base\_rate = reader[7].ToString(),

coefficient\_territory\_of\_use = reader[8].ToString(),

id\_coefficient\_inserance\_compensation = Convert.ToInt32(reader[9].ToString()),

id\_coefficient\_engine\_power = Convert.ToInt32(reader[10].ToString()),

id\_coefficient\_restriction\_use = Convert.ToInt32(reader[11].ToString()),

id\_coefficient\_experience\_age = Convert.ToInt32(reader[12].ToString()),

id\_coefficient\_period\_of\_use = Convert.ToInt32(reader[13].ToString()),

id\_coefficient\_insurance\_period = Convert.ToInt32(reader[14].ToString()),

price = reader[15].ToString()

}

);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка соединения с базой данных. \n\n\n\nПодрабнее:\n" + ex.ToString());

}

finally

{

dataBase2.closeConnection();

}

}

Функция MathButton – производит расчет ОСАГО.

private void MathButton(object sender, RoutedEventArgs e)

{

if (ComboWorker.SelectedItem != null & ComboStatusOsago.SelectedItem != null & ComboDriver.SelectedItem != null & ComboCar.SelectedItem != null & ComboPeriodUse.SelectedItem != null & ComboInsurancePeriod.SelectedItem != null & ComboRestriction.SelectedItem != null)

{

try

{

dataBase3.openConnection();

string query1 = $"select Car\_category.id\_car\_category , Car\_category.base\_rate from Car\_category,Car where Car\_category.id\_car\_category = ( select Car.id\_car\_category from Car where Car.id\_car = {car[ComboCar.SelectedIndex].id\_car})";

string query2 = $"select Region\_number.id\_region\_number , Region\_number.coefficient\_territory\_of\_use from Region\_number,Car where Region\_number.id\_region\_number = (select Car.id\_region\_number from Car where Car.id\_car = {car[ComboCar.SelectedIndex].id\_car})";

string query3 = $"select Coefficient\_insurance\_compensation.id\_coefficient\_insurance\_compensation, Coefficient\_insurance\_compensation.coefficient\_insurance\_compensation" +

$" from Coefficient\_insurance\_compensation,Driver where Coefficient\_insurance\_compensation.number\_insurance\_compensation = (select Driver.number\_insurance\_compensation from Driver where id\_driver = {driver[ComboDriver.SelectedIndex].id\_driver} ) and" +

$" Coefficient\_insurance\_compensation.coefficient\_beginning = (select Driver.coefficient\_beginning from Driver where id\_driver = {driver[ComboDriver.SelectedIndex].id\_driver} ) ";

string query4 = $"select Coefficient\_engine\_power.id\_coefficient\_engine\_power,Coefficient\_engine\_power.coefficient\_engine\_power from Coefficient\_engine\_power,Car where ( select Car.engine\_power from Car where Car.id\_car = {car[ComboCar.SelectedIndex].id\_car}) >= Coefficient\_engine\_power.engine\_power\_number ";

string query5 = $"select Driver.date\_of\_birth,Driver.driving\_experince from Driver where Driver.id\_driver = {driver[ComboDriver.SelectedIndex].id\_driver}";

SqlCommand cmd1 = new SqlCommand(query1, dataBase3.getConnection());

SqlDataReader SDA1 = cmd1.ExecuteReader();

while (SDA1.Read())

{

BaseBox.Text = SDA1[1].ToString();

}

SDA1.Close();

SqlCommand cmd2 = new SqlCommand(query2, dataBase3.getConnection());

SqlDataReader SDA2 = cmd2.ExecuteReader();

while (SDA2.Read())

{

RegionNumberBox.Text = SDA2[1].ToString();

}

SDA2.Close();

SqlCommand cmd3 = new SqlCommand(query3, dataBase3.getConnection());

SqlDataReader SDA3 = cmd3.ExecuteReader();

while (SDA3.Read())

{

InsuranceBox.Text = SDA3[0].ToString();

insurance = Convert.ToDouble(SDA3[1].ToString());

}

SDA3.Close();

SqlCommand cmd4 = new SqlCommand(query4, dataBase3.getConnection());

SqlDataReader SDA4 = cmd4.ExecuteReader();

if (SDA4.Read())

{

while (SDA4.Read())

{

PowerBox.Text = Convert.ToString(Convert.ToInt32(SDA4[0]));

enginepower = Convert.ToDouble(SDA4[1]);

}

}

else

{

PowerBox.Text = "1";

enginepower = 0.60;

}

SDA4.Close();

SqlCommand cmd5 = new SqlCommand(query5, dataBase3.getConnection());

SqlDataReader SDA5 = cmd5.ExecuteReader();

while (SDA5.Read())

{

var date2 = SDA5[0].ToString();

driving\_experince = Convert.ToInt32(SDA5[1]);

DateTime date1 = Convert.ToDateTime(date2);

if (date1 < DateTime.Today)

{

age = DateTime.Now.Year - date1.Year;

age++;

}

else

{

age = DateTime.Now.Year - date1.Year;

}

}

SDA5.Close();

string query6 = $"select id\_coefficient\_experience\_age,coefficient\_experience\_age from Coefficient\_experience\_age where age <= {age} and experience <= {driving\_experince} ";

SqlCommand cmd6 = new SqlCommand(query6, dataBase3.getConnection());

SqlDataReader SDA6 = cmd6.ExecuteReader();

while (SDA6.Read())

{

ExperienceBox.Text = SDA6[0].ToString();

expage = Convert.ToDouble(SDA6[1]);

}

double price = Convert.ToDouble(BaseBox.Text) \* expage \* enginepower \* insurance \* Convert.ToDouble(RegionNumberBox.Text) \* Convert.ToDouble(restriction[ComboRestriction.SelectedIndex].coefficient\_restriction\_use) \*

Convert.ToDouble(perioduse[ComboPeriodUse.SelectedIndex].coefficient\_period\_of\_use) \* Convert.ToDouble(insuranceper[ComboInsurancePeriod.SelectedIndex].coefficient\_insurance\_period);

PriceBox.Text = Math.Round(price, 2).ToString().Replace(",", ".");

PriceBox.Text = PriceBox.Text.Replace(",", ".");

BaseBox.Text = BaseBox.Text.Replace(",", ".");

RegionNumberBox.Text = RegionNumberBox.Text.Replace(",", ".");

if (StartBox.Text != "")

{

EndBox.Text = Convert.ToString(Convert.ToDateTime(StartBox.Text).AddMonths(Convert.ToInt32(insuranceper[ComboInsurancePeriod.SelectedIndex].insurance\_period)));

}

else

{

StartBox.Text = DateTime.Today.ToString();

EndBox.Text = Convert.ToString(DateTime.Today.AddMonths(Convert.ToInt32(insuranceper[ComboInsurancePeriod.SelectedIndex].insurance\_period)));

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка соединения с базой данных. \n\n\n\nПодрабнее:\n" + ex.ToString());

}

finally

{

dataBase3.closeConnection();

}

}

else

{

MessageBox.Show("Заполните Информацию полностью.");

}

}

Функция ReadyButton\_Click – производит изменение или добавление записи.

private void ReadyButton\_Click(object sender, RoutedEventArgs e)

{

if (ComboWorker.SelectedItem != null & ComboStatusOsago.SelectedItem != null & ComboDriver.SelectedItem != null & ComboCar.SelectedItem != null & ComboPeriodUse.SelectedItem != null & ComboInsurancePeriod.SelectedItem != null & ComboRestriction.SelectedItem != null & EndBox.Text != "" & BaseBox.Text != "")

{

if (countedit == 1)

{

try

{

string request = $"insert into Osago (id\_osago,id\_driver,id\_car,id\_worker,id\_status\_osago,start\_date,end\_date,base\_rate,coefficient\_territory\_of\_use,id\_coefficient\_inserance\_compensation,id\_coefficient\_engine\_power,id\_coefficient\_restriction\_use,id\_coefficient\_experience\_age,id\_coefficient\_period\_of\_use,id\_coefficient\_insurance\_period,price)" +

$" Values ('{count1 + 1}', '{driver[ComboDriver.SelectedIndex].id\_driver}', '{car[ComboCar.SelectedIndex].id\_car}', '{worker[ComboWorker.SelectedIndex].id\_worker}', '{statusosago[ComboStatusOsago.SelectedIndex].id\_status\_osago}', '{StartBox.Text}' , '{EndBox.Text}','{BaseBox.Text}', '{RegionNumberBox.Text}', '{InsuranceBox.Text}', '{PowerBox.Text}', '{restriction[ComboRestriction.SelectedIndex].id\_coefficient\_restriction\_use}'," +

$" '{ExperienceBox.Text}', '{perioduse[ComboPeriodUse.SelectedIndex].id\_coefficient\_period\_of\_use}', '{insuranceper[ComboInsurancePeriod.SelectedIndex].id\_coefficient\_insurance\_period}', '{PriceBox.Text}')";

dataBase3.openConnection();

SqlCommand sql = new SqlCommand(request, dataBase3.getConnection()); ;

sql.ExecuteNonQuery();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка соединения с базой данных. \n\n\n\n\nПодробнее:\n" + ex.ToString());

}

finally

{

dataBase3.closeConnection();

this.Close();

}

}

else

{

try

{

PriceBox.Text = PriceBox.Text.Replace(",", ".");

BaseBox.Text = BaseBox.Text.Replace(",", ".");

RegionNumberBox.Text = RegionNumberBox.Text.Replace(",", ".");

string request = $"update Osago set id\_driver = '{driver[ComboDriver.SelectedIndex].id\_driver}', id\_car = '{car[ComboCar.SelectedIndex].id\_car}', id\_worker = '{worker[ComboWorker.SelectedIndex].id\_worker}', " +

$" id\_status\_osago = '{statusosago[ComboStatusOsago.SelectedIndex].id\_status\_osago}', start\_date = '{StartBox.Text}', end\_date = '{EndBox.Text}',base\_rate = '{BaseBox.Text}' , coefficient\_territory\_of\_use = '{RegionNumberBox.Text}', id\_coefficient\_inserance\_compensation = '{InsuranceBox.Text}', id\_coefficient\_engine\_power = '{PowerBox.Text}' , id\_coefficient\_restriction\_use = '{restriction[ComboRestriction.SelectedIndex].id\_coefficient\_restriction\_use}', id\_coefficient\_experience\_age = '{ExperienceBox.Text}'," +

$"id\_coefficient\_period\_of\_use = '{perioduse[ComboPeriodUse.SelectedIndex].id\_coefficient\_period\_of\_use}',id\_coefficient\_insurance\_period = '{insuranceper[ComboInsurancePeriod.SelectedIndex].id\_coefficient\_insurance\_period}',price = '{PriceBox.Text}' where id\_driver = '{count1 + 1}'";

dataBase3.openConnection();

SqlCommand sql = new SqlCommand(request, dataBase3.getConnection()); ;

sql.ExecuteNonQuery();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка соединения с базой данных. \n\n\n\n\nПодробнее:\n" + ex.ToString());

}

finally

{

dataBase3.closeConnection();

this.Close();

}

}

}

else

{

MessageBox.Show("Заполните Информацию полностью.");

}

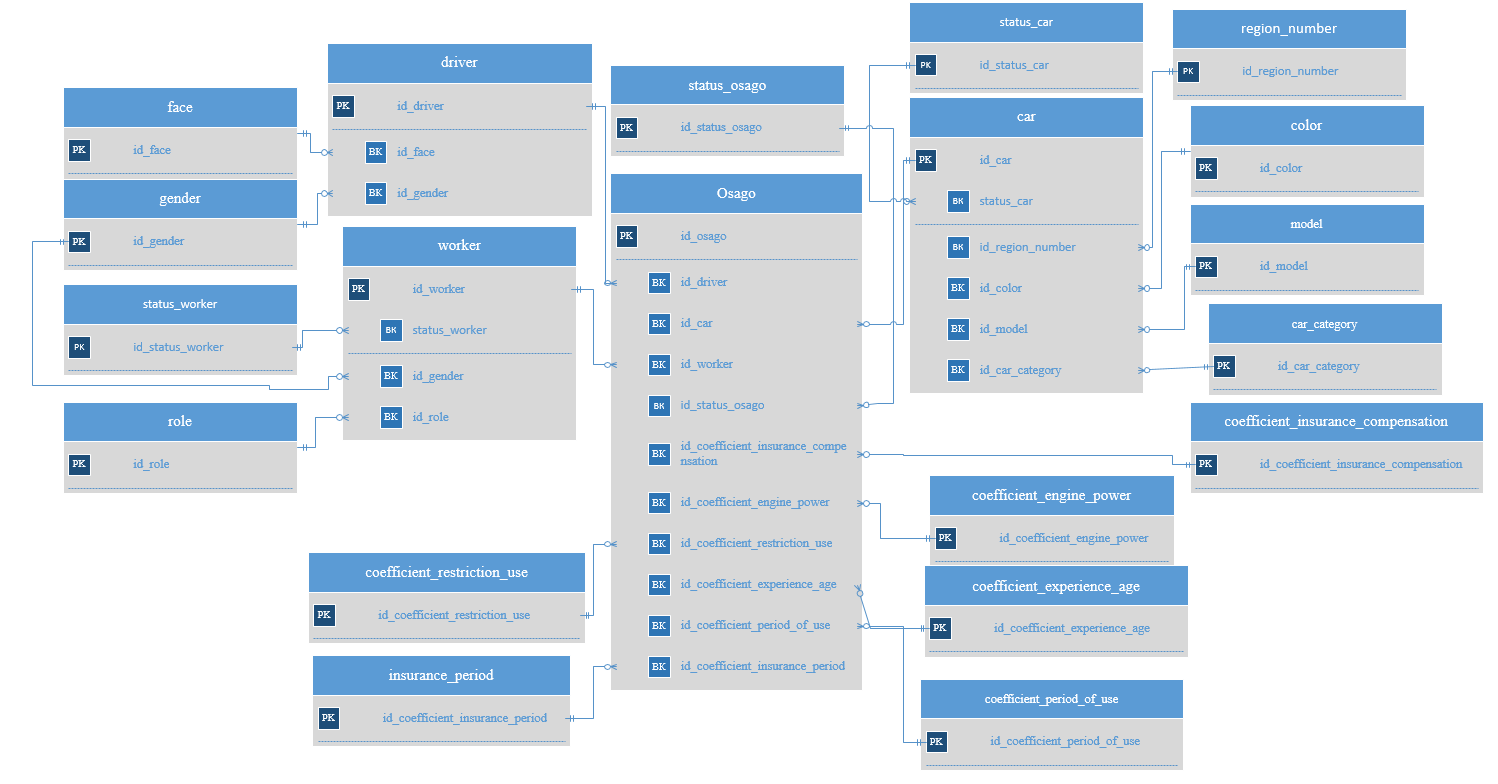
}

# **ПРИЛОЖЕНИЕ 2**

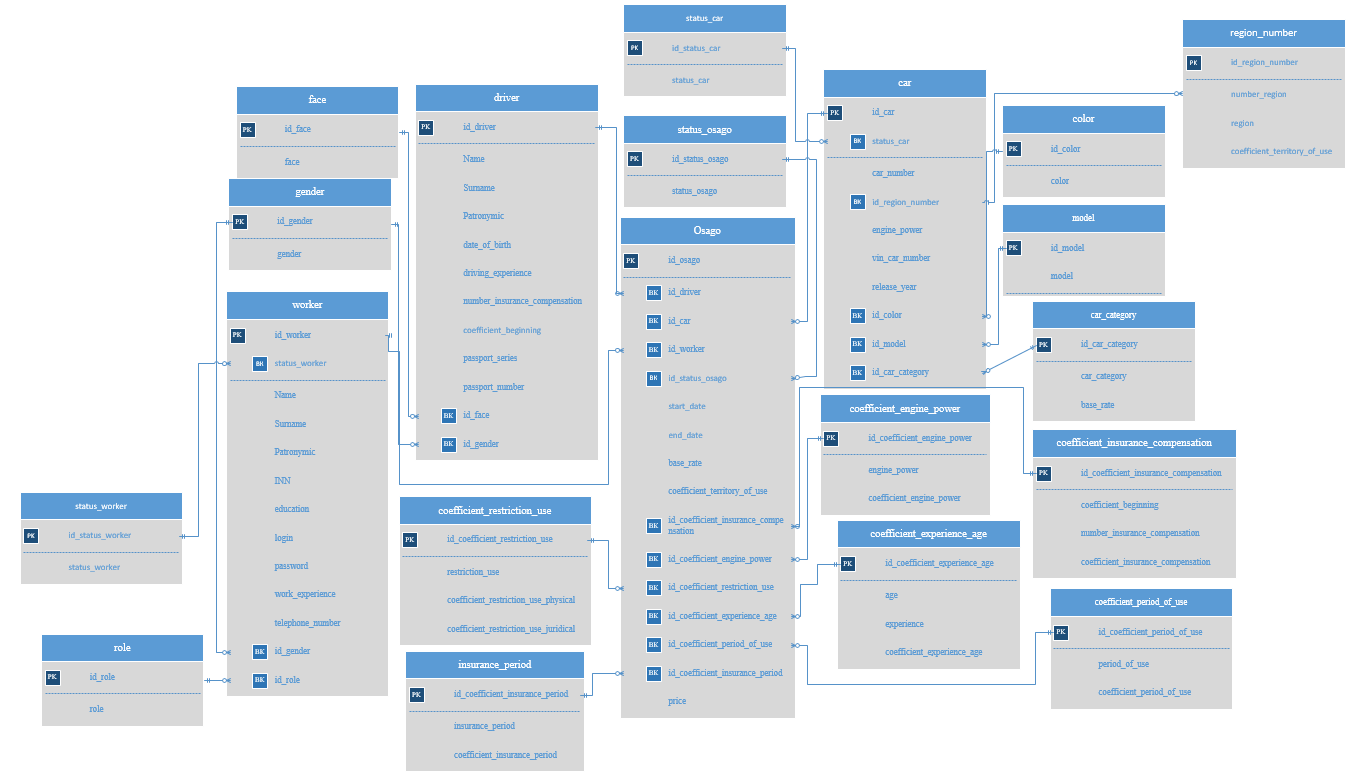
**Раздел проектирования**



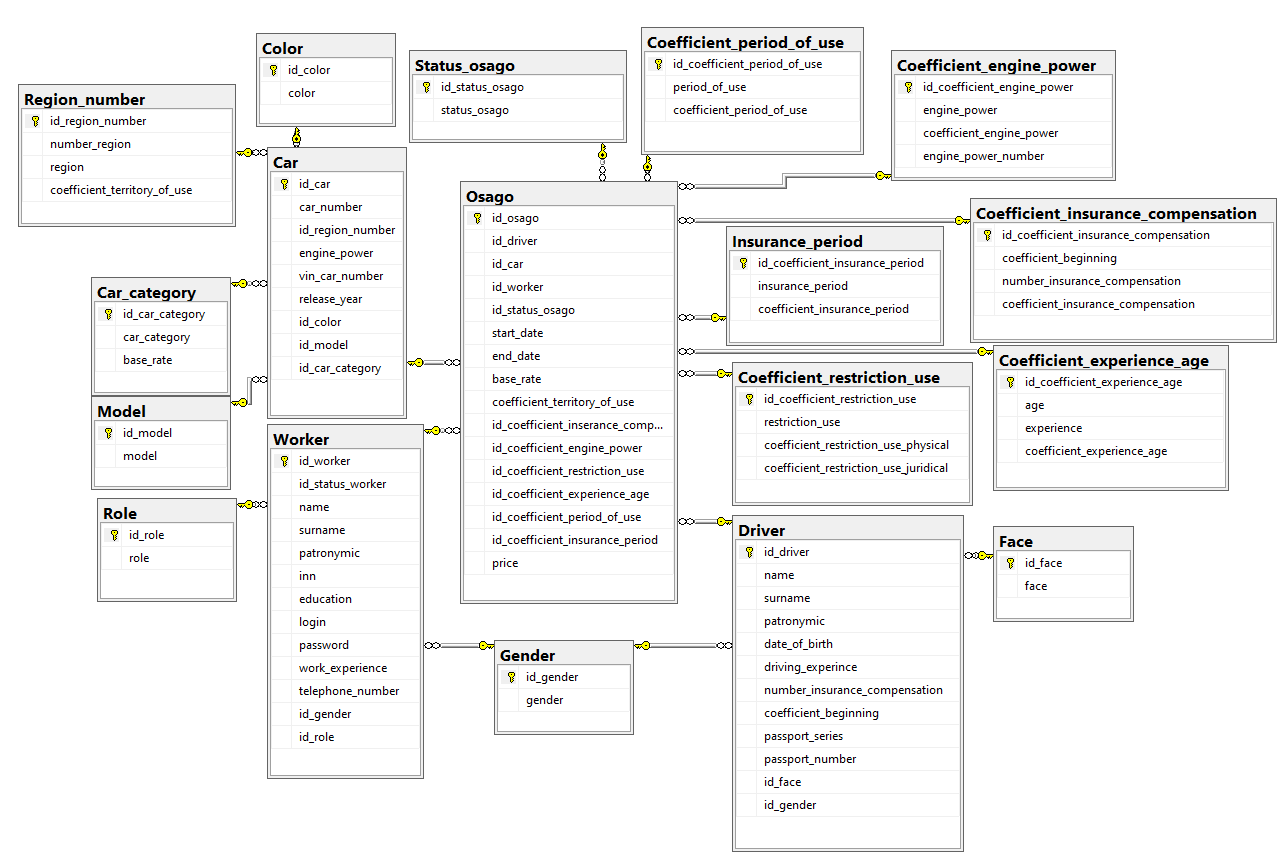
*Рисунок 1 - Use-case диаграмма*

****

*Рисунок 2 – Концептуальная модель БД*



*Рисунок 3 - Инфологическая модель БД*

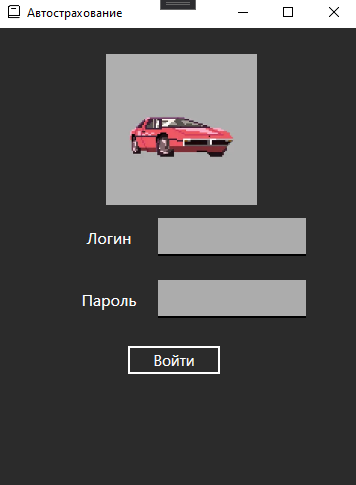


*Рисунок 4 -* Физическая модель БД

# **ПРИЛОЖЕНИЕ 3**

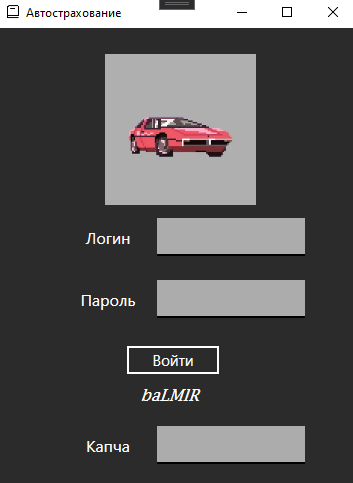
**Руководство пользователя**

Когда вы в первый раз запускаете программу вы попадаете на окно авторизации. При нажатии на кнопку, происходит авторизация. Смотреть Рисунок 1.



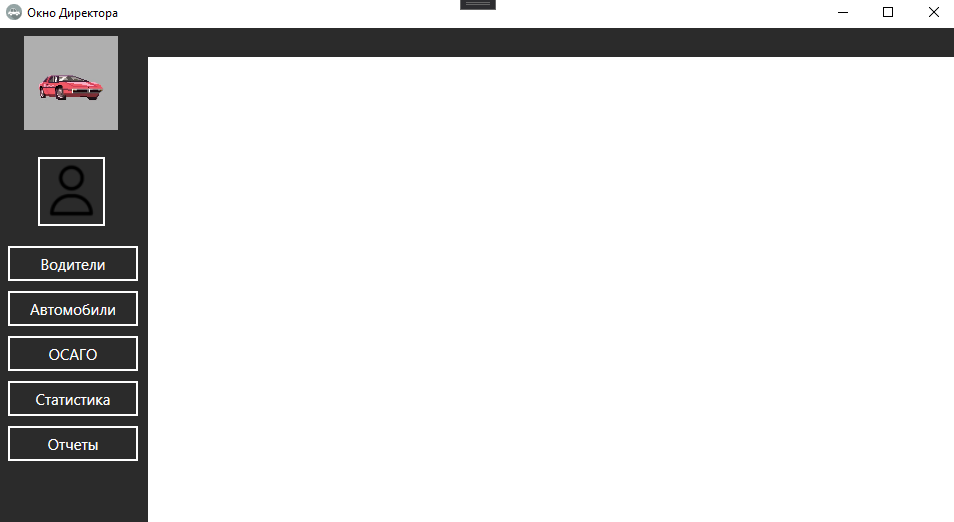
**Рисунок 1** – окно авторизации

В случае неправильного ввода логина и пароля в количестве 3 раз, приложение потребует ввести капчу. Смотреть Рисунок 2.



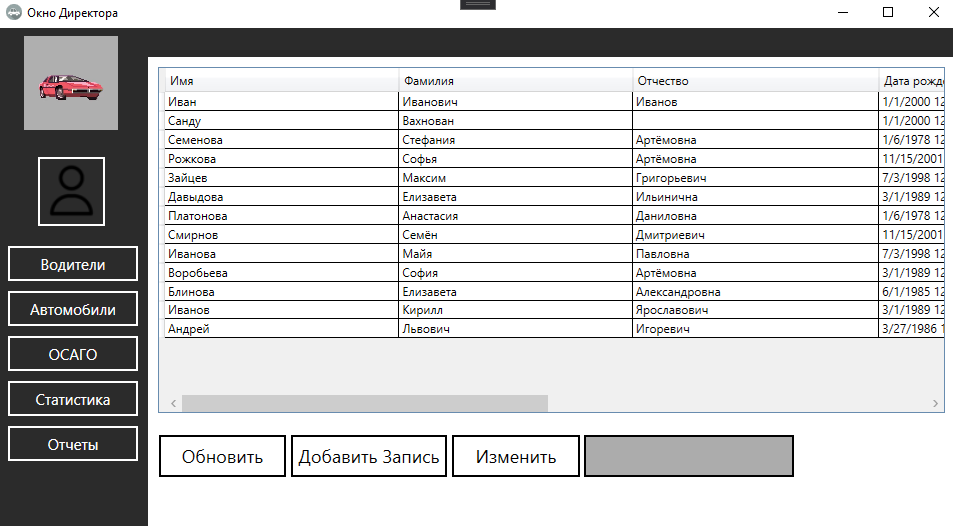
**Рисунок 2** – окно авторизации

После авторизации пользователь в зависимости от роли попадает на определенное окно. При нажатии на указанную кнопку происходит выбор фрейма.Смотреть Рисунок 3.



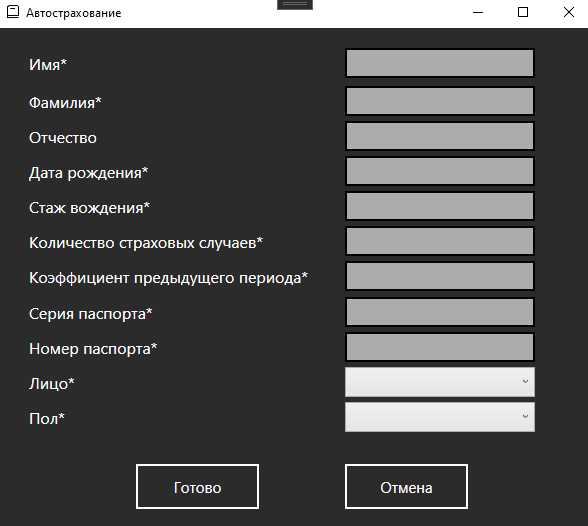
**Рисунок 3** – окно директора

После нажатия кнопки в левом меню фрейм появляется в окне фрейма, в данном случае фрейм клиентов. В данном фрейме присутствуют кнопки: Обновить, Добавить запись, Изменить и Искать. При нажатии на кнопку «Обновить» происходит обновление таблицы. При нажатии на кнопку «Искать» происходит поиск по столбцу Name. При нажатии на кнопку «Добавить запись» происходит переход на окно добавления записи. При нажатии кнопки «Изменить», если выбрана строчка таблицы, откроется окно изменения. Смотреть Рисунок 4.



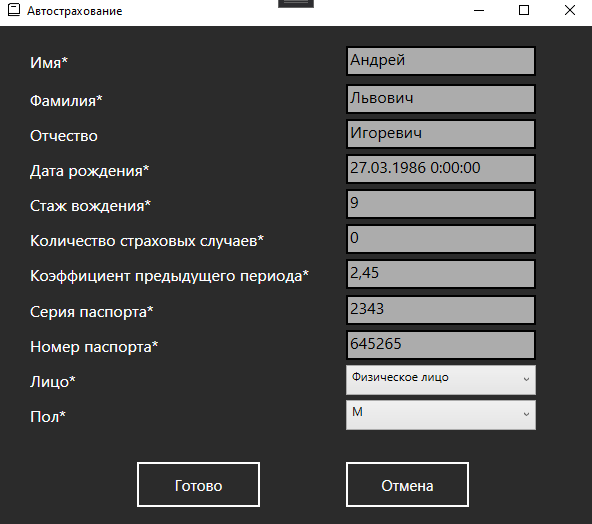
**Рисунок 4** – окно клиентов

После нажатия кнопки «Добавить запись» происходит открытие окна добавления. Смотреть Рисунок 5.



**Рисунок 5** – окно добавления

После заполнения обязательных полей, обозначенных знаком «\*», при нажатии на кнопку «Готово» происходит добавление записи. Смотреть Рисунок 6.



**Рисунок 6** – заполненное окно добавления

На окнах машин, ОСАГО, работников все работает по тем же принципам.

1. StatCounter [Электронный ресурс] - Режим доступа: <https://gs.statcounter.com/os-market-share/desktop/worldwide/#yearly-2021-2021-bar> [↑](#footnote-ref-1)
2. DB-engines [Электронный ресурс] - Режим доступа: https://db-engines.com/en/ranking [↑](#footnote-ref-2)
3. TIOBE [Электронный ресурс] - Режим доступа: https://tiobe.com/tiobe-index/ [↑](#footnote-ref-3)