**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 5

1. АНАЛИЗ ДАННЫХ 7

1.1. Описание предметной области 7

1.2. Постановка задачи 8

2. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ ЗАДАЧИ 10

2.1. Организация данных 10

2.2. Проектирование интерфейса программного средсва 15

3. РЕАЛИЗАЦИЯ 20

3.1. Структура проекта 20

3.2. Тестирование проекта 21

3.3. Спецификация проекта 22

4. ПРИМЕНЕНИЕ 23

4.1. Функциональное назначение 23

ЗАКЛЮЧЕНИЕ 25

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 26

Приложении A 27

Приложении Б 35

# ВВЕДЕНИЕ

Услуги такси сегодня пользуются большим спросом. Транспорт различного класса и назначения заказывают и приезжие, которые не знают местности, и коренные жители, которые хотят с комфортом добираться до нужного места.

Такси — средство общественного транспорта, обычно автомобиль, используемый для перевозки пассажиров и грузов в любую указанную точку с оплатой проезда машины по счётчику — таксометру. Иногда цена назначается по договору с водителем.[1]

Диспетчерская такси является предприятием-посредником. Это предприятие предоставляет перевозчикам информационные услуги и осуществляет координацию и связь между действиями пассажира, который вызвал такси, и водителя. Заказ бывает получен по телефону диспетчером, а затем водитель получает заказ по средствам радио.

Работа с данными происходит повсеместно. Для их хранения и обработки используются базы данных, а также приложения, позволяющие упросить работу с базами и данными в них.[2]

Проектируемая база данных предназначена для работников частного предприятия, оказывающего населению услуги такси, и позволяет им повысить эффективность своей работы за счет систематизации и быстрого поиска нужной им информации. Это упрощает работу, т.к. отпадает необходимость просматривать горы бумаг в поисках нужной информации, рассчитывать стоимость путевого листа каждому водителю и отчеты о работе администрации.

Среди аналогов можно выделить «О-Такси», «Такси- Мастер» и другие системы автоматизации службы такси, которые распространяются на платной основе и несут обширный функционал. Высокие цены и сложности настройки являются причиной поиска других решений.

Целью курсового проектирования является разработка приложения, реализующего хранение и обработку данных с возможностью фильтрации данных по ключам и формировании отчетов. Приложение должно позволять регистрироваться и входить под учетной записью пользователя, позволять пользователю хранить, вводить и редактировать его данные, а также фильтровать уже имеющиеся данные, формировать отчеты.

Для реализации поставленной цели необходимо решить следующие задачи:

изучить предметную область, осуществить обзор литературных источников;

изучить имеющиеся аналоги и проанализировать их;

спроектировать функционал;

спроектировать внешний вид интерфейса;

протестировать работу программы;

оформить пояснительную записку;

Первый раздел «Анализ данных» состоит из описания предметной области и постановки задачи.

Второй раздел «Объектно-ориентированное проектирование задачи» состоит из организации данных, проектирования интерфейса программного средства, выбора и обоснования среды разработки.

В третьем разделе «Реализация» представлены структура проекта, тестирование проекта и спецификации проекта.

Четвертый раздел «Применение» служит для описания функционального назначения, технических требований и инсталляции.

В заключении подведены итоги проделанной работы.

# 1. АНАЛИЗ ДАННЫХ

## 1.1. Описание предметной области

Работа такси осуществляется следующим образом: каждый водитель, заступая на смену, связывается с диспетчером, и тот вносит его в карточку работающих в данный момент. Сведения карточки отображаются на экране. Водители держат обратную связь по радио.

Заказы поступают по телефону к диспетчеру, он записывает необходимые данные в базу заказов. Дата и время поступления заказа диспетчер вводит сам или вводиться автоматически.[3]

Клиент может сразу у диспетчера узнать стоимость заказа, и только потом заказывать.

Потом из списка свободных водителей выбирает того, кто будет выполнять заказ или водитель сам отвечает по обратной связи, что примет заказ. После выполнения заказа водитель по рации отчитывается перед диспетчером.

После каждой смены диспетчер формирует отчет, который показывает, сколько заказов поступило, сколько выполнилось, сколько было отменено, и их общую стоимость.

Эти отчеты поступают к администратору, на основании этого всего администратор формирует общий отчет (сводный отчет) за определенную дату.

В конце месяца администратор, пользуясь своими отчетами, формирует отчет по итогам месяца и отправляет его высшему руководству такси.

Еще администратор занимается кадрами. Он формирует дела новых сотрудников, и вносит необходимые поправки в дела уже работающих на этом предприятии. На администраторе лежит ответственность за правильное формирование дел и их сохранность.

Проблемы, которые могут возникнуть при осуществлении данной деятельности такие:

1) Потеря данных диспетчером, или неправильная их трактовка;

2) Выбор не оптимального маршрута следования таксистов;

3) Не корректная форма представления отчетов и т.д.

В курсовой работе в соответствии с заданием автоматизируется деятельность организации «Такси»

Основными видами деятельности являются:

Обеспечение безопасности перевозок;

Обеспечение комфортабельности поездки;

Своевременное выполнение заявок;

Одной из основных проблем на предприятии является отслеживание состояния приема и выполнения заявок и отсутствие полной базы данных на Водителей и диспетчеров, т.е. при поступлении заказа необходимо выводить полную информацию о нем и готовность на данный момент.

Для решения этих проблем необходимо своевременно получать информацию о заявках и своевременно оповещать водителей об адресе клиента.

## 1.2. Постановка задачи

Поводом для автоматизации организации Такси послужило необходимость в данной подсистеме, и облегчения ежедневной работы, т.к. эта работа слишком трудоемкая и не защищена от неточностей и ошибок обусловленным человеческим фактором. Для облегчения и ускорения работы, было принято решение создать базу данных, которая обеспечивает:

-Надежность и безопасность;

-Автоматизацию документооборота;

-Быстрое составление отчетов.

Автоматизация этих функций призвана, увеличить скорость и качество обработки информации.

Автоматизированная информационная система в данной предметной области поможет нам упростить работу такси, убрать заполнение документов вручную, ускорить обслуживание клиентов.

Автоматизированная информационная система в данной предметной области упростит работу такси, позволит автоматически генерировать отчеты.

Основные цели, для достижения которых создано приложение:

Обеспечение работникам более быстрого и удобного поиска необходимой информации;

Обеспечение порядка размещения уже хранящихся и поступающих данных;

Тщательное отслеживание изменений данных;

Обеспечение большей защиты информации от несанкционированного доступа;

Осуществление учета заказов.

Программа должна обеспечивать безопасное хранение, иметь удобный графический интерфейс для просмотра, добавления, редактирования и удаления информации. Формировать отчеты.

Программа должна наглядно и просто демонстрировать все данные, изменять их, удалять их без потери другой информации, добавлять новые данные. Интерфейс должен быть предельно удобным и делать приложение простым в применении, что в свою очередь делает возможным использовании приложения пользователями с разными уровнями знания компьютерной техники, даже человеком, который владеет только основами знаний по информатике.

В итоге приложение должно иметь возможности:

Просмотра информации;

Добавления информации;

Удаление информации;

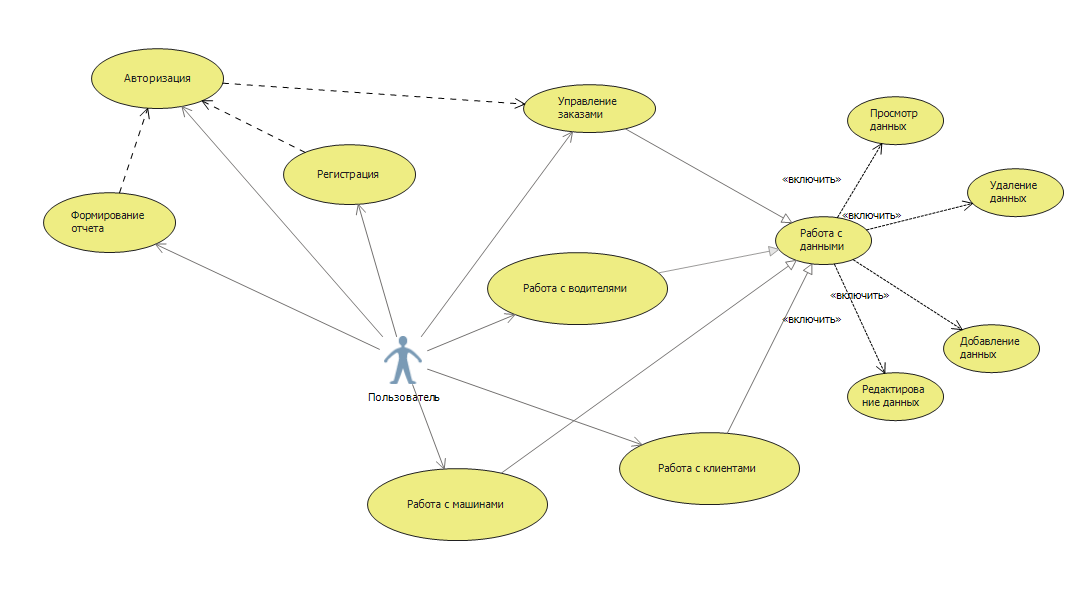
Редактирование информации;

Формирование отчета;

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования действующих лиц, участвующими в процессе.

Варианты использования являются описаниями типичных взаимодействий между пользователями системы и самой системой. Они отображают внешний интерфейс системы и указывают форму того, что система должна сделать.

На основе выше сказанного была построена следующая диаграмма вариантов использования Рис. 1.2.1.



**Рисунок 1.2.1 –диаграмма вариантов использования, описывающая взаимодействие с приложением.**

**Источник: собственная разработка**

# 2. ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОЕКТИРОВАНИЕ ЗАДАЧИ

## 2.1. Организация данных

Для проектирования базы данных в приложении используется MS SQL Server в Visual Studio 2015.[4]

Для работы с базой данных используется объектно-ориентированная технология доступа к данным Entity Framework с подходом code-first.

Выходными данными, в разрабатываемом приложении, являются поля из базы данных, которая хранится на сервере.

Для данного приложения была разработана реляционная база данных.

База данных состоит из одиннадцати таблиц, связанных типом один ко многим, многие ко многим.

При исследовании предметной области были выделены следующие сущности:

«Order/Заказы» - содержит основную информацию по принятым заказам;

«Application\_user/Пользователь программы» - содержит информацию по зарегистрированным пользователям;

«Driver/Водители» - содержит информацию о водителях;

«Client/Клиенты» - содержит информацию о клиентах;

«Car/Автомобили» - содержит информацию об автомобилях;

«Discount/Скидки» - содержит информацию о скидках;

«Status/Статус» - содержит информацию о состоянии заказа;

«Brand/Марка» - содержит информацию о марках автомобиля;

«Type\_car/Тип машины» - содержит информацию о типе автомобиля;

«Street/Улицы» - содержит информацию о типе улицах;

«District/Дистрикт» - содержит информацию о районах;

Описание атрибутов сущностей.

Атрибуты сущности «Order»:

Id- уникальный идентификатор записи;

Phone\_number - телефонный номер заказчика;

Id\_client – идентификационный номер клиента;

Id\_status – идентификационный номер статуса заказа;

Data\_order – дата/время заказа;

Start\_value – начальное значения таксометра;

End\_value – конечное значение таксометра;

Id\_application\_user – идентификатор номер пользователя приложения;

Id\_street\_from – идентификатор номер начальной улицы.

Id\_street\_to – идентификатор номер улицы назначения.

Id\_driver - идентификационный номер водителя.

Атрибуты сущности «Application\_user»:

Id- уникальный идентификатор записи;

Email - email пользователя;

Email\_confirmed – наличие подтверждения email;

Password\_hash - хэш значения пароля;

Phonr\_number – телефонный номер;

Phonr\_number\_confirmed – наличие подтверждения телефонного номера;

User\_name – имя/логин пользователя.

Атрибуты сущности «Street»:

Id- уникальный идентификатор записи;

Name –название улицы;

Id\_district – идентификационный номер района.

Атрибуты сущности «District»:

Id- уникальный идентификатор записи;

Name –название района.

Атрибуты сущности «Car»:

Id- уникальный идентификатор записи;

Name –название машины.

Id\_brand – идентификационный номер марки автомобили.

Id\_type - идентификационный номер типа автомобили.

Place – количество мест в машине;

Gos\_number –гос номер машины;

Stars – комфорт машины;

Active – состояние машины.

Атрибуты сущности «Brand»:

Id- уникальный идентификатор записи;

Type – марка машины;

Атрибуты сущности «Type\_car»:

Id- уникальный идентификатор записи;

Type – тип машины;

Атрибуты сущности «Driver»:

Id- уникальный идентификатор записи;

First\_name – имя водителя;

Second\_name – фамилия водителя;

Patronymic – отчество водителя;

Phone\_number – телефонный номер;

Date\_of\_birth – дата рождения;

Data\_of\_employment –дата принятия на работу;

Category –категория водителя;

Id\_district – расположение водителя;

Id\_car – автомобиль водителя.

Атрибуты сущности «Client»:

Id- уникальный идентификатор записи;

First\_name – имя клиента;

Second\_name – фамилия клиента;

Patronymic – отчество клиента;

Phone\_number – телефонный номер;

Id\_discount – идентификационный номер скидки.

Атрибуты сущности «Discount»:

Id- уникальный идентификатор записи;

Name – название скидки;

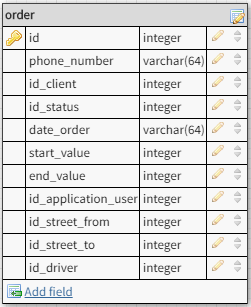
Discount\_percent – процент скидки.

Атрибуты сущности «Status»:

Id- уникальный идентификатор записи;

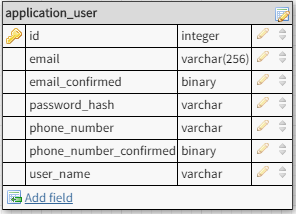
Status – статус заказа.

Структуры таблиц разработанной БД приведены на рисунках 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7, 2.8, 2.9, 2.10, 2.11.



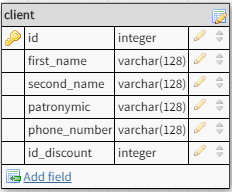
**Рисунок 2.1 – Структура таблицы “Order”.**

**Источник: собственная разработка**



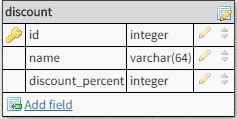
**Рисунок 2.2 – Структура таблицы “Application\_user”.**

**Источник: собственная разработка**



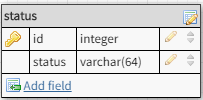
**Рисунок 2.3 – Структура таблицы “Client”.**

**Источник: собственная разработка**



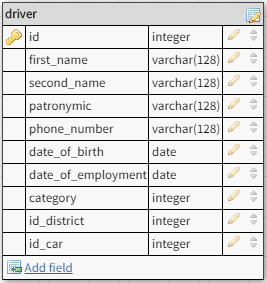
**Рисунок 2.4 – Структура таблицы “Discount”.**

**Источник: собственная разработка**



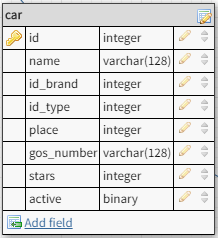
**Рисунок 2.5 – Структура таблицы “Status”.**

**Источник: собственная разработка**



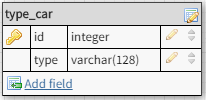
**Рисунок 2.6 – Структура таблицы “Driver”.**

**Источник: собственная разработка**



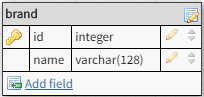
**Рисунок 2.7 – Структура таблицы “Car”.**

**Источник: собственная разработка**



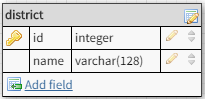
**Рисунок 2.8 – Структура таблицы “Type\_car”.**

**Источник: собственная разработка**



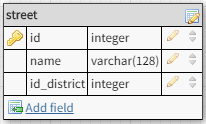
**Рисунок 2.9 – Структура таблицы “Brand”.**

**Источник: собственная разработка**



**Рисунок 2.10 – Структура таблицы “District”.**

**Источник: собственная разработка**



**Рисунок 2.11 – Структура таблицы “District”.**

Источник: собственная разработка

На основе этих данных была построена схема базы данных, которая  представлена в Приложении A.

## 2.2. Проектирование интерфейса программного средсва

В современном мире миллиарды вычислительных устройств. Еще больше программ для них. И у каждого своего интерфейса, являющийся «рычагами» взаимодействия между пользователем и машинным кодом. Не удивительно, что чем лучше интерфейс, тем эффективнее взаимодействие.

Проектирование дизайна подразделяется на 6 этапов.

Этап I. Предпроектный аналия

Работы по проектированию интерфейса начинаются с предпроектного анализа. На рабочей сессии с клиентом мы описываем видение проекта (vision), в котором рассказывается о его сути и целях, а также перечисляем предполагаемую функциональность системы в виде кратких сценариев взаимодействия. В дополнение к этому проводится анализ потребностей и контекста работы целевой аудитории, которая описывается в виде ключевых персонажей. Также составляется первоначальная карта сайта, которая показывает примерную структуру будущей системы. На написание и утверждение этих базовых документов обычно уходит около 3 дней, после чего мы планируем остальные работы и даем точную оценку сроков и стоимости их выполнения. Поэтому удобнее вести предпроектный анализ по отдельному договору — сложно получить точную оценку двухмесячной работы без предварительного исследования.

Этап II. Сбор требований

На следующем этапе мы готовим подробный перечень функциональности (user stories). Он позволяет учесть все функциональные требования и лучше понять особенности будущей системы. На его основе мы делаем вывод, какие из функций требуют целого процесса, какие — просто отдельной страницы, а каким будет достаточно простой кнопки. Ориентируясь на составленных ранее персонажей, мы обновляем карту сайта и составляем схему навигации. После этого рисуются диаграммы переходов между страницами — они объединяют страницы системы в рамках конкретных процессов. Теперь мы знаем, как пользователи будут работать с продуктом в целом и как именно выполнять конкретные задачи. Этап длится около 4 дней.

Этап III. Проектирование интерфейса

Третий этап — самый важный. Здесь мы создаем структурные схемы страниц (wireframes), которые показывают, какая информация и элементы управления должны располагаться на страницах системы. Это еще не дизайн, но уже основа для него — wireframes являются техническим заданием для дизайнера. Общение с клиентом на этом этапе достаточно плотное — уточнение вопросов и утверждение чертежей идет по нескольку раз в день. Но и результатов хватает — в зависимости от сложности проекта выходит от нескольких десятков до пары сотен схем страниц. Длительность этапа — от одной до нескольких недель.

Этап IV. Дизайн интерфейса

Завершающим этапом становится визуальный дизайн интерфейса. Сперва на основе пары ключевых страниц мы отрабатываем креативную концепцию. После того как общая стилистика одобрена клиентом, отрисовываются дизайн-макеты ключевых страниц системы. На этом этапе продукт обретает внешний вид — до этого мы занимались его сутью и принципами работы. Для проектов, которые планируют активно развиваться, мы также готовим руководство по стилю интерфейса (style guide). Он описывает принципы визуального оформления продукта и позволит сохранить его целостность в процессе доработок. Работы по этому этапу продолжаются 1–2 недели.

Этап V. Подготовка спецификации

При необходимости мы готовим предварительное техническое задание на разработку системы. Оно объединяет в себе полученные ранее документы, расширяет и перечисляет дополнительные требования к системе — функциональные, архитектурные, эксплуатационные. По желанию клиента могут быть составлены подробные сценарии взаимодействия, которые пошагово описывают процесс работы пользователя с системой.

Финальный этап. Приемка

Приемка работ клиентом может проходить одним большим пакетом замечаний или разбиваться на несколько более мелкие этапы. Сроки, в которые замечания должны быть выставлены, оценены и исправлены оговариваются в договоре.[8]

Для объектного моделирования был использован язык UML. UML ( Unified Modeling Language) унифицированный язык моделирования) язык графического описания для объектного моделирования в областиразработки программного обеспечения, моделирования бизнеспроцессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UMLмоделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UMLмоделей возможна генерация кода [4].

UML позволяет также разработчикам программного обеспечения достигнуть соглашения в графических обозначениях для представления общих понятий (таких как класс, компонент, обобщение (англ. generalization), агрегация (англ. aggregation) и поведение) и больше сконцентрироваться на проектировании и архитектуре.[6]

Преимущества UML

UML объектно-ориентирован, в результате чего методы описания результатов анализа и проектирования семантически близки к методам программирования на современных объектно-ориентированных языках;

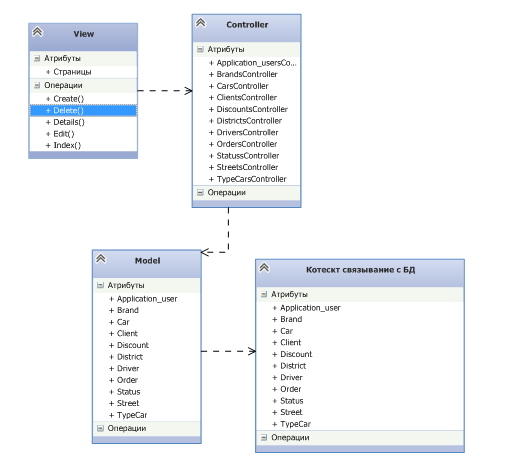
UML позволяет описать систему практически со всех возможных точек зрения и разные аспекты поведения системы;

Диаграммы UML сравнительно просты для чтения после достаточно быстрого ознакомления с его синтаксисом;

UML расширяет и позволяет вводить собственные текстовые и графические стереотипы, что способствует его применению не только в сфере программной инженерии;

UML получил широкое распространение и динамично развивается.

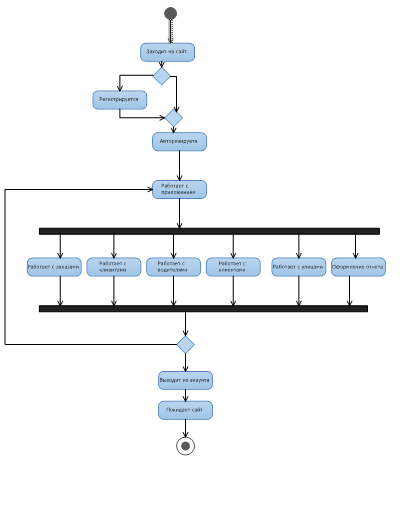
Для того чтобы показать статическую структуру системы (т.е. определить типы объектов и статические связи между ними) была разработана диаграмма классов, которая представлена на рисунке 2.2.1.



**Рисунок 2.2.1 - UML диаграмма классов.**

**Источник: собственная разработка**

Что бы отобразить последовательность действия пользователя при взаимодействии с приложением была разработана диаграмма деятельности, которая изображена на рисунке 2.2.2.



**Рисунок 2.2.2 – UML диаграмма деятельности**

**Источник: собственная разработка**

2.3 Выбор и обоснование среды разработки

Для решения задачи была выбрана интегрированная среда разработки Microsoft Visual Studio 2015.

Visual Studio – интегрированная среда разработки от компании Microsoft. Visual Studio используется для разработки консольных приложений, приложений с графическим интерфейсом, веб сайтов, веб приложений, программ с поддержкой Windows Forms, а также для работы с системой построения клиентских приложений - Windows Presentation Foundation (WPF). Можно разрабатывать веб-сервисы и в родном и в управляемом кодах для всех платформ, поддерживаемых .NET Framework, Microsoft Windows, Windows Mobile, Microsoft Silverlight и .NET Compact Framework.[9]

Редактор кода в Visual Studio позволяет использовать такие языки, как C#, C++, VB.NET, HTML, CSS, JavaScript, XAML, SQL и многие другие. Встроенный отладчик может работать как отладчик на уровне исходного кода, так и как отладчик машинного уровня. Встраиваемые инструменты включают в себя визуальные редакторы форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. В Visual Studio можно создавать и подключать сторонние плагины для расширения функциональности и добавления новых наборов инструментов.[9]

Visual Studio выгодно отличается эффективностью и надежностью. Кроме того, Visual Studio является интегрированной средой, в которой разработчики могут использовать уже имеющиеся навыки для написания кода, отладки, модульного тестирования и развертывания непрерывно расширяющейся номенклатуры типов приложений. Visual Studio упрощает реализацию общих задач и обеспечивает индивидуальный подход, что позволяет разработчикам максимально использовать возможности базовых платформ.

Мощные инструменты Visual Studio помогают быстро создавать высококачественный код. Интегрированная поддержка разработки через тестирование и новые инструменты отладки программ для многоядерных процессоров позволяют без труда находить и устранять дефекты, обеспечивая создание высококачественного решения.[9]

Авторы C# стремились создать язык, сочетающий простоту и выразительность современных объектно-ориентированных языков c богатством возможностей и мощью C++. По словам Андерса Хейлсберга, C# позаимствовал большинство своих синтаксических конструкций из C++. В частности, в нем присутствуют такие удобные типы данных, как структуры и перечисления. Синтаксические конструкции С# унаследованы не только от C++, но и от Visual Basic. Например, в С#, как и в Visual Basic, используются свойства классов. Как C++, С# позволяет производить перегрузку операторов для созданных вами типов. С# - это фактически гибрид разных языков. При этом этот язык синтаксически не менее (если не более) чист, чем Java, так же прост, как Visual Basic, и обладает практически той же мощью и гибкостью, что и C++.

Особенности С#:

полный и хорошо определенный набор основных типов;

встроенная поддержка автоматической генерации XML-документации. Автоматическое освобождение динамически распределенной памяти;

возможность отметки классов и методов атрибутами, определяемыми пользователем. Это может быть полезно при документировании и способно воздействовать на процесс компиляции (например, можно пометить методы, которые должны компилироваться только в отладочном режиме);

полный доступ к библиотеке базовых классов .NET, а также легкий доступ Windows-API;

указатели и прямой доступ к памяти, если они необходимы. Однако язык разработан таким образом, что практически во всех случаях можно обойтись и без этого;

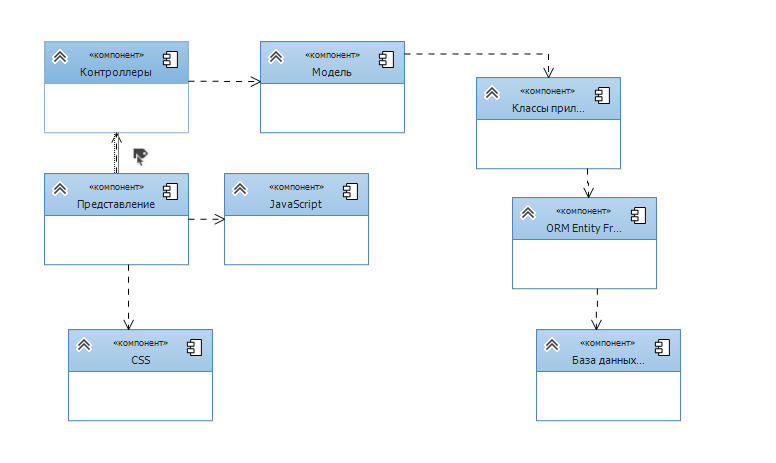
поддержка свойств и событий в стиле VB. Простое изменение ключей компиляции. Позволяет получать исполняемые файлы или библиотеки компонентов .NET, которые могут быть вызваны другим кодом так же, как элементы управления ActiveX.(компоненты.СОМ);

возможность использования С# для написания динамических web-страниц ASP.NET.[13]

# 3. РЕАЛИЗАЦИЯ

## 3.1. Структура проекта

Структура проекта представлена с помощью диаграммы компонентов изображенной на рисунке 3.1.



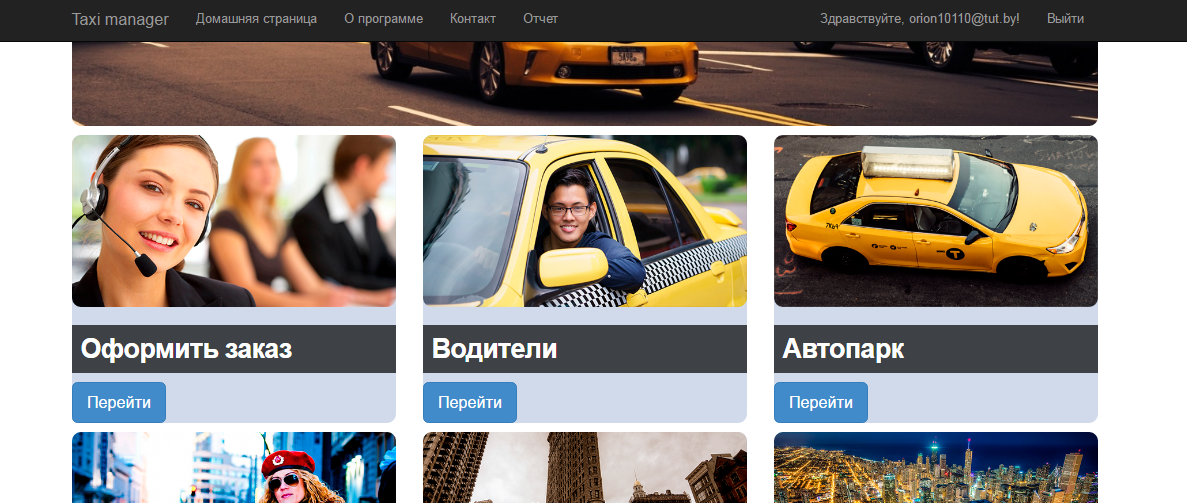
**Рисунок 3.1 - Структура таблицы “Поставщики”**

**Источник: собственная разработка**

На этой диаграмме расположены основные компоненты приложения. Контекст данных, связи данных с моделью, контроллерами и представлением. Зависимости представления от CSS и JS, это показано при помощи вида связи - “Зависимость” [10].

Одно из важнейших качеств хорошего приложения - удобный интерфейс, созданный с учетом всех требований к пользовательскому интерфейсу.

Интерфейс приложения изображен на рисунке 3.2.



**Рисунок 3.2 - Пользовательский интерфейс приложения**

**Источник: собственная разработка**

## 3.2. Тестирование проекта

Одним из основных методов оценки качества ПО является тестирование. Тестирование программного средства - это процесс выполнения программ на некотором наборе данных, для которого заранее известен результат применения или известны правила поведения этих программ. Указанный набор данных называется тестовым или просто тестом. Тестирование программ является одной из составных частей более общего понятия - «отладка программ». Под отладкой понимается процесс, позволяющий получить программу, функционирующую с требующимися характеристиками в заданной области изменения входных данных.[10]

Тестирование имеет две различные цели:

продемонстрировать разработчикам и заказчикам, что программа соответствует требованиям;

выявить ситуации, в которых поведение программы является неправильным, нежелательным или не соответствующим спецификации.

В зависимости от доступа разработчика тестов к исходному коду тестируемой программы различают «тестирование белого ящика» и «тестирование чёрного ящика».

Тестирование проводилось сначала по принципу «белого ящика», а затем «черного ящика».

При тестировании белого ящика, разработчик теста имеет доступ к исходному коду программ и может писать код, который связан с библиотеками тестируемого программного обеспечения. Это типично для модульного тестирования, при котором тестируются только отдельные части системы. Оно обеспечивает то, что компоненты конструкции — работоспособны и устойчивы, до определённой степени. При тестировании белого ящика используются метрики покрытия кода или мутационное тестирование.[11]

При тестировании чёрного ящика, тестировщик имеет доступ к программе только через те же интерфейсы, что и заказчик или пользователь, либо через внешние интерфейсы, позволяющие другому компьютеру либо другому процессу подключиться к системе для тестирования. Например, тестирующий модуль может виртуально нажимать клавиши или кнопки мыши в тестируемой программе с помощью механизма взаимодействия процессов, с уверенностью в том, все ли идёт правильно, что эти события вызывают тот же отклик, что и реальные нажатия клавиш и кнопок мыши. Обычно в данном виде тестирования критерий покрытия складывается из покрытия структуры входных данных, покрытия требований и покрытия модели (в тестировании на основе моделей).[11]

На основе технического задания был создан чек-лист.

Юзабилити (usability) тестирование – это определение степени комфортности и удобства для пользователей.

Тестирование юзабилити:

удобство расположения полей, кнопок на форме приложения;

точно и грамотно ли названы элементы на форме;

много ли ошибок пользователь допускает при работе с приложением и легко ли исправляются эти ошибки;

Проверка валидности входных данных:

контроль корректности входных данных;

контроль выдачи сообщений пользователю при обнаружении ошибок во входных данных и предпринимаемые действия при обработке ошибок;

реакция программного средства на некорректно введенные данные.

Все функции приложения, такие как обращение к базе данных и манипуляции с пользовательским интерфейсом, были протестированы и отлажены, набор основных тестов для приложения изображён в таблице 3.1.

|  |  |
| --- | --- |
| Проверка | Ожидаемый результат |
| Проверить вывод данных | Отображение данных |
| Добавление записей в таблицах | Новые записи в таблицах |
| Изменение записей в таблицах | Измененные данные записей в таблице |
| Удаление записей в таблицах | Указанные записи удалены из таблиц |
| Выполнить фильтрацию | Отображение данных с указанными фильтрами |
| Выполнить авторизацию | Предоставление функций авторизированного пользователя |

Таблица 3.1 – Таблица тесты приложения

## 3.3. Спецификация проекта

Программный продукт, со всем своим содержимым (изображения, БД, решение) имеет размер 218 мб.

Спецификация и наименование файлов проекта, представлены в таблице 3.2.

|  |  |
| --- | --- |
| Обозначение: | Наименование: |
| TestTaxi.sln | Файл решения |
| .png | Изображения используемые в оформлении проекта. |
| aspnet-TestTaxi-20161014124335.mdf | База данных проекта. |
| .cshtml | Файлы разметки. |
| .cs | Файлы классов. |
| Web.config | Конфигурационный файл для приложения |

Таблица 3.2 – Спецификация и наименование файлов проекта.

# 4. ПРИМЕНЕНИЕ

## 4.1. Функциональное назначение

Программное средство разработано с целью упрощения и повышения качества работы менеджера такси.

Данный продукт будет использоваться менеджерами таксопарка.

Для проектирования данного приложения использовались следующие case-средства:

Microsoft Visual Studio 2015;

Microsoft SQL Server 2016

DbDesigner.net.

Интерфейс приложения легко понятен пользователю. Интерфейс лёгок и запоминаем, эффективен и надёжен в использовании. При этом достаточно простой и состоит из довольно очевидных и элементарных типов действий, которые, так или иначе, ведут пользователя к намеченной цели.

Разграничения доступа присутствует.

Пользователям предоставлена возможность вставлять и удалять записи из таблиц.

Доступна функция поиска по базе данных.

Данное приложение является серверным.

4.2. Технические требования

На сервере должна быть установлен ISS сервер.

Требование ISS сервера:

ОЗУ – 8ГБ;

Процессор - 64-разрядный четырехъядерный;

Место на жестком диске - 80 ГБ для системного диска.

На стороне клиента должен быть установлен любой браузер, версии не ниже, чем в приведенной ниже спецификации для самых популярных браузеров:

IE - 9;

Google Chrome - 48;

Opera - 19;

FireFox - 38.

4.3. Инсталляция

Как правило, большинство веб-приложений ASP.NET публикуются на веб-сервер IIS.

Конфигурируем веб-сервер. Для этого нужно открыть средство администрирования IIS: зайти в Панель управления, затем выбрать Администрирование-Диспетчер служб IIS.

Далее необходимо перейти в каталог веб-узла и создать в нем каталог после чего необходимо в контекстном меню папки выбрать добавить приложение.

В появившемся окне ввести соответствующие настройки (в качестве физического пути приложения созданный выше каталог).

Далее необходимо в каталоге разместить приложение.

Перейдем к приложению в Visual Studio. Нажмем правой кнопкой на название проекта и в появившемся меню выберем Publish.

Перед нами откроется мастер публикации, который предложит нам пройти несколько этапов. В начале выберем профиль. Если не одного профиля не определено, то создадим, нажав на ссылку New... и выбрав какое-нибудь название.

После создания профиля нажмем на Next и перейдем к следующему этапу - Connection. На этом этапе для опции Publish Method выберем File System

Для опции Target Location определим физический путь к каталогу нашего сайта. В данном случае это путь C:\inetpub\wwwroot\TaxiManager.

А для поля Destination URL указываем url, по которому будет доступно приложение, а именно http://localhost/ TaxiManager.

Теперь запустим Службу веб-публикаций через консоль Служб или через консоль IIS и можем обращаться к сайту по url http://localhost/BookStore.

# ЗАКЛЮЧЕНИЕ

В результате проделанной работы разработано было разработано приложения для упрощения работы менеджера таксопарка. Также были созданы UML диаграммы: вариантов использования, деятельности, классов, последовательности и компонентов.

Программа создана в интегрированной среде разработки Microsoft Visual Studio 2015 с использованием базы данных MS SQL Server 2016.

В процессе разработки были применены навыки работы с бэкэндом и фронэндом.

Кроме того, были получены знания об особенностях работы серверных приложении.

Данное приложение может быть более подробно и тщательно доработано в будущем, что принесет его разработчику немалую прибыль.

Таким образом, задание на курсовой проект выполнено в полном объеме. Разработанный проект полностью удовлетворяет требованиям заказчика и находится на стадии внедрения.

Данный курсовой проект является примером того, что с развитием техники и программирования многие сферы деятельности, где требовалось заполнение вручную большого количества документов теперь можно автоматизировать. Разработка и внедрение проектов такого типа доступна практически в любой сфере жизнедеятельности человека и даст, конечно же, возможность организовать работу с учетом требований сегодняшнего времени.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Информация о службе такси [Режим доступа]: https://ru.wikipedia.org/wiki/Такси. [Дата доступа]: 10.10.2016

Учёт заказов на услуги такси. [Режим доступа]: http://biglib.info/22018-uchet-zakazov-na-uslugy-taksy.html. [Дата доступа]: 11.10.2016

Информационный портал о такси. [Режим доступа]: http://taxi.contact-centr.ru/. [Дата доступа]: 09.06.2016

Действующее лицо (actor). [Режим доступа]: www. studopedia.org/11-2748.html. [Дата доступа]: 15.10.2016

5. Структуры в Си-шарп. [Режим доступа]: www.mycsharp.ru/post/48/2014\_12\_21\_struktury\_v\_si-sharp.html. [Дата доступа]: 14.10.2016

6. Простота пользовательского интерфейса. Описание.  
[Режим доступа]: www. http://it-kniga.narod.ru/5-7502-0222-4/02010201.html. [Дата доступа]: 16.10.2016

7. Диаграмма деятельности и особенности ее построения.   
[Режим доступа]: www.intuit.ru/studies/courses/32/32/lecture/1020. [Дата доступа]: 18.10.2016

8. Основные понятия диаграмм классов UML.   
[Режим доступа]: www.citforum.ru/database/advanced\_intro/31.shtml. [Дата доступа]: 10.10.2016

9. Шилдт Герберт C# 4.0. Полное руководство. – Вильямс, 2015. – 1056 c.

10. Тестирование и отладка программного средства.   
[Режим доступа]: www.bourabai.kz/alg/pro10.htm. [Дата доступа]: 20.10.2016

11. Тестирование программного обеспечения.   
[Режим доступа]: www.dpgrup.ru/testing.htm. [Дата доступа]: 22.10.2016

12. Microsoft Visual Studio. Средства разработки приложений.   
[Режим доступа]: www. msdn.microsoft.com/ru-ru/library/dn762121.aspx. [Дата доступа]: 19.10.2016

13. История создания языка программирования C#. [Режим доступа]: www. interestprograms.ru/articles/historyprogramming/historycsharp. [Дата доступа]: 15.10.2016

# Приложении A

**Листинг программы**

Код модели Order.

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Web.Mvc;

namespace TestTaxi.Models

{

public class Order

{

[HiddenInput(DisplayValue = false)]

public int Id { get; set; }

[Display(Name = "Статус")]

public Status status { get; set; }

[Display(Name = "Телефон")]

public string PhoneNumber { get; set; }

[Display(Name = "Дата заказа")]

public DateTime DateOrder { get; set; }

[HiddenInput(DisplayValue = false)]

public int? ClientID { get; set; }

[HiddenInput(DisplayValue = false)]

public int? DriverID { get; set; }

[Display(Name = "Клиент")]

public virtual Client Client { get; set; }

[Display(Name = "Водитель")]

public virtual Driver Driver { get; set; }

[HiddenInput(DisplayValue = false)]

public int? StreetFromID { get; set; }

[HiddenInput(DisplayValue = false)]

public int? StreetToID { get; set; }

[Display(Name = "Улица от")]

public virtual Street StreetFrom { get; set; }

[Display(Name = "Улица куда")]

public virtual Street StreetTo { get; set; }

[Display(Name = "Начальное значение")]

public int? StartValue { get; set; }

[Display(Name = "Конечное значение")]

public int? EndValue { get; set; }

[HiddenInput(DisplayValue = false)]

public string ApplicationUserID { get; set; }

}

}

Код контролера OrdersController

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Net;

using System.Web;

using System.Web.Mvc;

using TestTaxi.Models;

using Microsoft.AspNet.Identity;

namespace TestTaxi.Controllers

{

[Authorize]

public class OrdersController : Controller

{

private ApplicationDbContext db = new ApplicationDbContext();

// GET: Orders

public ActionResult Index(int page = 1,int status = 0)

{

//List<Status> listBrand = new List<Status>() { Status.ВСЕ, Status.ЗАВЕРШЕН, Status.ОБРАБАТЫВАЕТСЯ, Status.ПРИНЯТ };

//SelectList status = new SelectList(listBrand, "Status", "Status");

int pageSize = 10;

string id = User.Identity.GetUserId();

IEnumerable<Order> orderPerPages = db.Orders.Where(s=>s.ApplicationUserID == id).Include(o => o.Client).Include(o => o.Driver).Include(o => o.StreetFrom).Include(o => o.StreetTo)

.OrderBy(p => p.DateOrder).Skip((page - 1) \*

pageSize).Take(pageSize);

PageInfo pageInfo = new PageInfo

{

PageNumber = page,

PageSize = pageSize,

TotalItems = db.Orders.Count()

};

MyIndexViewModel<Order> ivm = new MyIndexViewModel<Order>

{

PageInfo = pageInfo,

Keeps = orderPerPages

};

return View(ivm);

}

// GET: Orders/Details/5

public ActionResult Details(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Order order = db.Orders.Find(id);

if (order == null)

{

return HttpNotFound();

}

return View(order);

}

// GET: Orders/Create

public ActionResult Create()

{

ViewBag.ClientID = new SelectList(db.Clients, "Id", "FirstName");

ViewBag.DriverID = new SelectList(db.Drivers, "Id", "FirstName");

ViewBag.StreetFromID = new SelectList(db.Streets, "Id", "Name");

ViewBag.StreetToID = new SelectList(db.Streets, "Id", "Name");

return View();

}

// POST: Orders/Create

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку. Дополнительные

// сведения см. в статье http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create([Bind(Include = "Id,status,PhoneNumber,DateOrder,ClientID,DriverID,StreetFromID,StreetToID,StartValue,EndValue")] Order order)

{

string idUser= User.Identity.GetUserId();

order.ApplicationUserID = idUser;

if (ModelState.IsValid)

{

db.Orders.Add(order);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.ClientID = new SelectList(db.Clients, "Id", "SecondName", order.ClientID);

ViewBag.DriverID = new SelectList(db.Drivers, "Id", "SecondName", order.DriverID);

ViewBag.StreetFromID = new SelectList(db.Streets, "Id", "Name", order.StreetFromID);

ViewBag.StreetToID = new SelectList(db.Streets, "Id", "Name", order.StreetToID);

return View(order);

}

// GET: Orders/Edit/5

public ActionResult Edit(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Order order = db.Orders.Find(id);

if (order == null)

{

return HttpNotFound();

}

ViewBag.ClientID = new SelectList(db.Clients, "Id", "SecondName", order.ClientID);

ViewBag.DriverID = new SelectList(db.Drivers, "Id", "SecondName", order.DriverID);

ViewBag.StreetFromID = new SelectList(db.Streets, "Id", "Name", order.StreetFromID);

ViewBag.StreetToID = new SelectList(db.Streets, "Id", "Name", order.StreetToID);

return View(order);

}

// POST: Orders/Edit/5

// Чтобы защититься от атак чрезмерной передачи данных, включите определенные свойства, для которых следует установить привязку. Дополнительные

// сведения см. в статье http://go.microsoft.com/fwlink/?LinkId=317598.

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit([Bind(Include = "Id,status,PhoneNumber,DateOrder,ClientID,DriverID,StreetFromID,StreetToID,StartValue,EndValue")] Order order)

{

string idUser = User.Identity.GetUserId();

order.ApplicationUserID = idUser;

if (ModelState.IsValid)

{

db.Entry(order).State = EntityState.Modified;

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.ClientID = new SelectList(db.Clients, "Id", "FirstName", order.ClientID);

ViewBag.DriverID = new SelectList(db.Drivers, "Id", "FirstName", order.DriverID);

ViewBag.StreetFromID = new SelectList(db.Streets, "Id", "Name", order.StreetFromID);

ViewBag.StreetToID = new SelectList(db.Streets, "Id", "Name", order.StreetToID);

return View(order);

}

// GET: Orders/Delete/5

public ActionResult Delete(int? id)

{

if (id == null)

{

return new HttpStatusCodeResult(HttpStatusCode.BadRequest);

}

Order order = db.Orders.Find(id);

if (order == null)

{

return HttpNotFound();

}

return View(order);

}

// POST: Orders/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

Order order = db.Orders.Find(id);

db.Orders.Remove(order);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

db.Dispose();

}

base.Dispose(disposing);

}

}

}

Код предсавление Order – Index

@model TestTaxi.Models.MyIndexViewModel<TestTaxi.Models.Order>

@using TestTaxi.Helpers

@{

ViewBag.Title = "Заказы";

}

<h2>Заказы</h2>

<p>

@Html.ActionLink("Добавить", "Create")

</p>

<table class="table">

<tr>

<th>

Клиент

</th>

<th>

Водитель

</th>

<th>

Улица от

</th>

<th>

Улица куда

</th>

<th>

Статус

</th>

<th>

Телефон

</th>

<th>

Дата заказа

</th>

<th>

Значение таксометра

</th>

<th>

Значение таксометра после

</th>

<th></th>

</tr>

@foreach (var item in Model.Keeps) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Client.SecondName)

</td>

<td>

@Html.DisplayFor(modelItem => item.Driver.SecondName)

</td>

<td>

@Html.DisplayFor(modelItem => item.StreetFrom.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.StreetTo.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.status)

</td>

<td>

@Html.DisplayFor(modelItem => item.PhoneNumber)

</td>

<td>

@Html.DisplayFor(modelItem => item.DateOrder)

</td>

<td>

@Html.DisplayFor(modelItem => item.StartValue)

</td>

<td>

@Html.DisplayFor(modelItem => item.EndValue)

</td>

@\*<td>

@Html.DisplayFor(modelItem => item.ApplicationUserID)

</td>\*@

<td>

@Html.ActionLink("Изменить", "Edit", new { id=item.Id }) |

@Html.ActionLink("Детали", "Details", new { id=item.Id }) |

@Html.ActionLink("Удалить", "Delete", new { id=item.Id })

</td>

</tr>

}

</table>

<div class="btn-group">

@Html.PageLinks(Model.PageInfo, x => Url.Action("Index", new { page = x }))

</div>

# Приложении Б

**Схема базы данных**

