Національний аерокосмічний університет ім. М.Є. Жуковського

“Харківський авіаційний інститут”

Факультет економіки та менеджменту

Кафедра інженерії програмного забезпечення

КУРСОВИЙ ПРОЕКТ

з курсу **«ПЗ корпоративних інформаційних систем»**

ХАІ.603.651п.16В.05010301.126376.ПЗС

Тема: «Гостиничный бизнес: учет предоставления дополнительных услуг (бар, пляж, салон красоты, дискотека)»

Студента \_\_5\_\_ курсу \_\_651п\_\_ групи

напряму підготовки **Програмне забезпечення систем**

\_\_\_\_Чорнобривця А.Б.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(прізвище та ініціали)

Керівник \_\_\_ Дегтярьова Т. Г. \_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_ ст. виклад.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Національна шкала \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількість балів: \_\_\_\_\_\_Оцінка: ECTS \_\_\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис) (прізвище та ініціали

м. Харків – 2016 рік

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc448440355)

[1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 6](#_Toc448440356)

[1.1 Автоматизированная система комплексных расчетов учета предоставления дополнительных услуг 6](#_Toc448440357)

[1.2 СУБД 8](#_Toc448440358)

[1.3 Основные термины 8](#_Toc448440359)

[2 ЗАДАЧИ И НАЗНАЧЕНИЕ ПО 10](#_Toc448440360)

[3 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ 11](#_Toc448440361)

[3.1 Техническое обеспечение 11](#_Toc448440362)

[3.2 Компьютер пользователя 11](#_Toc448440363)

[3.3 Информационное обеспечение 11](#_Toc448440364)

[3.4 Программное обеспечение 12](#_Toc448440365)

[4 ТЕХНИЧЕСКОЕ ЗАДАНИЕ 13](#_Toc448440366)

[5 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ ПОДСИСТЕМЫ УЧЕТА ПРЕДОСТАВЛЕНИЯ ДОПОЛНИТЕЛЬНЫХ УСЛУГ 19](#_Toc448440367)

[5.1 Требования заказчика 19](#_Toc448440368)

[5.1.1 Мандатные требования 19](#_Toc448440369)

[5.1.2 Ограничительные требования 20](#_Toc448440370)

[5.1.3 Требования ПО 20](#_Toc448440371)

[5.1.4 Построение диаграммы вариантов использования 22](#_Toc448440372)

[6 КОНЦЕПТУАЛЬНЫЕ ПРОЕКТИРОВАНИЯ 24](#_Toc448440373)

[6.1 Характеристика предприятия и анализ существующей АИР системы 24](#_Toc448440374)

[6.2 Функциональная модель IDEF0 25](#_Toc448440375)

[6.3 Диаграммы потоков данных (Data Flow Diagramming) 29](#_Toc448440376)

[7 ПРОЕКТИРОВАНИЕ БД 31](#_Toc448440377)

[7.1 Логическое проектирование 31](#_Toc448440378)

[7.2 Описание физической базы данных 33](#_Toc448440379)

[7.3 Построение физической модели БД 35](#_Toc448440380)

[7.4 Построение запросов физической модели данных 35](#_Toc448440381)

[8 ОБОСНОВАНИЕ ВЫБОРА СУБД И ИНСТРУМЕНТАЛЬНОЙ СРЕДЫ РАЗРАБОТКИ 36](#_Toc448440382)

[9 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПОДСИСТЕМЫ УЧЕТА ПРЕДОСТАВЛЕНИЯ ДОПОЛНИТЕЛЬНЫХ УСЛУГ 38](#_Toc448440383)

[9.1 Диаграмма классов 38](#_Toc448440384)

[9.2 Описание диаграммы классов 38](#_Toc448440385)

[9.3 Листинги 40](#_Toc448440386)

[9.4 Скриншоты экранных работ 40](#_Toc448440387)

[10 ТЕСТИРОВАНИЕ ПО 45](#_Toc448440388)

[10.1 Планы интеграционного тестирования 45](#_Toc448440389)

[10.2 Планы системного тестирования 48](#_Toc448440390)

[10.3 Результаты интеграционного тестирования 49](#_Toc448440391)

[10.4 Результаты выполнения системного тестирования 50](#_Toc448440392)

[ВЫВОДЫ 52](#_Toc448440393)

[СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ 53](#_Toc448440394)

[ПРИЛОЖЕНИЕ А 54](#_Toc448440395)

[ПРИЛОЖЕНИЕ Б 55](#_Toc448440396)

[ПРИЛОЖЕНИЕ В 56](#_Toc448440397)

[ПРИЛОЖЕНИЕ Г 75](#_Toc448440398)

# ВВЕДЕНИЕ

В настоящее время в связи с развитием компьютерной техники появилась возможность автоматизировать многие процессы, с одной стороны. С другой стороны, в связи со стремительным развитием интернета и комплекса предоставляемых услуг увеличился объем обрабатываемой информации. И возникла объективная необходимость автоматизировать корпоративную систему учета предоставления дополнительных услуг (бар, пляж, салон красоты, дискотека).

Современная гостиница представляет собой сложный комплекс функциональных звеньев, от слаженности работы которого зависит успешность существования предприятия на рынке. Учитывая современные тенденции в сфере гостеприимства и конкуренцию, что усиливается, повышается необходимость обеспечения оперативности и точности работы персонала и гостиничного комплекса в целом. Решение данной проблемы возможно лишь за счет внедрения систем автоматизации работы отеля, то есть внедрение корпоративных автоматизированных систем управления (КАСУ) отелем [1].

Автоматизированные системы управления для гостиничных комплексов является комплексом интегрированных подсистем, которые создают эффективную среду взаимодействия сотрудников, клиентов и деловых партнеров — туристических агентств, корпоративных клиентов и туроператоров. И хотя цена таких систем высока, согласно исследованиям корпорации Microsoft, большинство отелей на Западе (особенно сетевых) периодически устанавливают новую систему управления. Это вызвано темпами роста конкурентной борьбы и технологического прогресса — если раньше отели меняли техническое оснащение в среднем каждые 7-9 лет, то сегодня — каждые 3-5 лет, и тенденция сокращения этого срока сохраняется.

Наиболее популярными на этом рынке являются западные системы — «Micros Fidelio», «Lodging Touch LIBICA», Hospitality Enterprise Resource Planning «Cenium», Epitome PMS, Amadeus PMS, OPERA; российские разработки — “Nimeta», «Эдельвейс», «Отель», «KEI Hotel», «UCS Shelter», соответствующие модули корпоративных систем Галактика и Парус [1].

Для гостиниц вопросы комплексной автоматизации процессов бронирования мест расчетов с гостями, агентами и туроператорами, проблемы безопасности объекта и хозяйственной деятельности становятся все более актуальными [2].

Целью данной курсовой работы является создание автоматизированной системы учета предоставления дополнительных услуг (бар, пляж, салон красоты, дискотека). Также разработать автоматизированную систему расчета за услуги, предоставляемые организациями, которая автоматизирует процессы регистрации своих операторов, учета использования услуг, выполнения расчетов стоимости предоставленных каждому заказа услуг, печати счетов и других выходных документов, приема, регистрации и контроля оплаты услуг, предоставленных посетителям, сбора статистических данных и их анализа.

Для этого необходимо создать базы данных, содержащих сведения о клиентах, их персональных данных и предоставляемых дополнительных услуг: бар, пляж, салон красоты, дискотека.

Достижение цели осуществляется посредством комплекса задач:

* проектирование и создание таблиц для хранения данных;
* ввод данных;
* разработка других элементов базы, предназначенных для просмотра, редактирования и вывода информации.

Курсовой проект преследует цель формирования умений и навыков разработки клиент - серверных приложений, предназначенных для автоматизации комплексных расчетов за услуги, полученные ими теоретические знания, приобретенные навыки самостоятельной работы, умение синтезировать полученные знания и применять их к решению практических задач.

Выполнение курсовой работы является одним из завершающих и наиболее значимым этапом обучения студентов по дисциплине «Программное обеспечение корпоративных информационных систем».

Конечной целью курсовой работы является выявление уровня подготовки студента, степени готовности его к усвоению профессиональных знаний [3].

Основными задачами выполнения КП является:

* выявление понимания студентом основных проблем и перспектив развития технологии проектирования автоматизированных информационных систем;
* выявление понимания студентом значимости своей будущей профессиональной деятельности, умения приобретать новые знания, особенно в области современных информационных технологий;
* выявление умения работать с технической и нормативной документацией, а также четко излагать свои мысли;
* выявление навыков решать поставленные практические задачи с использованием теоретических знаний [3].

# 1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 1.1 Автоматизированная система комплексных расчетов учета предоставления дополнительных услуг

*Автоматизированная система* комплексных *расчетов* учета предоставления дополнительных услуг за услуги предназначена для автоматизации процессов обслуживания физических и юридических лиц и расчетов с ними за все виды услуг.

Современные АСР реализуют компьютерную технологию обработки информации, легко адаптируется к потребностям конкретного потребителя (предприятиям связи, коммунальных услуг, провайдерам Интернет, больницам и др.) и могут быть использованы в организациях как самостоятельно функционирующая система, так и в составе эксплуатируемой АСУ.

Объектами автоматизации при внедрении АСР являются основные отделы и службы организаций, к которым относятся:

* расчетный отдел;
* информационно-вычислительный центр;
* пункты приема платежей;
* бухгалтерия;
* справочная служба;
* отдел АСУ.

Основными концепциями, используемыми при создании АСР, являются: глобальность, целостность, конфиденциальность, доступность, живучесть, массштабируемость и открытость, которые обеспечиваются за счет использования в АСР современных информационных архитектур, технологических платформ и инструментальных средств.

*Глобальность системы* определяется тем, что вся информация, входящая в состав системы, рассматривается как неделимая целостность.

*Доступность* определяется параллельной работой всех служб организаций в режиме ⌠on-line], вносящих соответствующие изменения в содержание информации базы данных.

Устойчивость и целостность определяются высоким качеством применяемого программного обеспечения (ПО), а также использованием необходимого для этих вопросов организационного обеспечения, связанного с архивацией данных, их достоверностью и т.п.

*Конфиденциальность* обеспечивается тем, что доступ всех пользователей АСР к данным осуществляется только в пределах установленных для каждого работника полномочий.

*Массштабируемость* и *открытость* обеспечиваются использованием при создании АСР современных инструментальных средств программирования, которые не накладывают ограничений на размеры баз данных предприятия, объем документооборота, емкость сети и число рабочих мест в системе.

Принципы построения АСР опираются на использование современных технологических платформ, топологий и архитектур информационных систем, обеспечивающих высокую степень безопасности, надежности, целостности и сохранности информации, а также быстрое выполнение транзакций при работе с базой данных (БД).

АСР реализуют в соответствии с архитектурой «Клиент-сервер» с возможностью подключения удаленных локальных вычислительных сетей (LAN), реализованных по архитектуре «Файл-сервер».

"Клиенты АСР, представляющие собой автоматизированные рабочие места (АРМ), реализованы на технологических платформах WINDOWS ХХ , UNIX. При этом "клиенты" АСР имеют следующие свойства:

* удаленный доступ ко всем ресурсам АСР;
* определение доступа пользователя к услугам в зависимости от класса его авторизации;
* одинаковый пользовательский графический интерфейс для всех услуг.

Обычно АСР ориентирована на организацию информационного обмена между "Клиентами" и "Сервером", а также между серверами с помощью корпоративной вычислительной сети. При этом в системе имеется возможность удаленного доступа к ресурсам АСР и информационного обмена с другими автоматизированными системами по каналам связи, в том числе и коммутируемым, посредством использования протокола ТСР/IР или другого.

АСР обладает возможностью использования как "бумажной", так и "безбумажной" технологии обработки информации с использованием режимов "on-line" и/или "off-line". При этом рабочие места АСР подключаются к серверу базы данных посредством вычислительной сети, коммутируемых и некоммутируемых каналов связи, по которым осуществляется постоянная актуализация БД.

Максимальное удаление автоматизированных рабочих мест АСР от сервера без использования модема определяется конфигурацией сети и ограничениями, которые накладываются стандартами.

Информирование абонентов об их задолженности или сумме требуемой оплаты осуществляется путем распечатки почтовых сообщений и их рассылки, а также путем распечатки и передачи в пункты оплаты платежных реестров или файлов, содержащих аналогичную информацию.

Автоматизация деятельности организации при использовании АСР предусматривает автоматизацию следующих функций, выполняемых оператором заказов:

* просмотр предоставляемых услуг;
* ведение картотеки пользователей;
* ввод исходной информации за услуги, предоставленные пользователю за расчетный период;
* тарификация услуг, предоставленных пользователям;
* тарификация номеров, предоставленных пользователям;
* расчет и проведение начислений по концу расчетного периода;
* формирование необходимых форм итоговых расчетных, статистических и платежных документов;
* формирование по начислениям за любой период времени;
* прием и учет платежей клиентов за предоставленные им услуги.

Автоматизация функций организаций, предоставляемых услуги обеспечивается следующими подсистемами:

* Подсистема контроля и ввода информации.
* Подсистема административного обслуживания пользователей.
* Подсистема расчетов.
* Подсистема обработки статистики.

Доступ пользователей АСР к системе регламентируется и регулируется "Администратором" системы, присваивающим пользователям системы определенные имена и пароли, уровни полномочий доступа к однозначно определенным функциям, объектам и данным, и закрепляющим эти функции за конкретным пользователем АСР

Вход оператора АСР в систему осуществляется после ввода его имени и пароля, контроля правильности имени и пароля и его регистрации, в результате чего на экране монитора рабочего места пользователя появляется соответствующее меню системы.

## 1.2 СУБД

База данных (БД) – это совокупность специальным образом организованных данных хранимых в памяти вычислительной системы отображающих состояние объектов и их взаимосвязи в рассматриваемой предметной области.

Предметной областью называется часть реальной системы, представляющая интерес для определенного исследования. Различать полную предметную область (предприятие) и организационную единицу этой предметной области (отделы).

## 1.3 Основные термины

Объект – термин, обозначающий факт, лицо, событие, предмет, о котором могут быть собраны данные (каждая таблица представляет один объект);

Реляционная БД - основной тип современных баз данных. Состоит из таблиц, между которыми могут существовать связи по ключевым значениям.

Таблица базы данных (table) – регулярная структура, которая состоит из однотипных строк (записей, records), разбитых на столбцы (поля, fields).

В теории реляционных баз данных синоним таблицы – отношение (relation), в котором строка называется кортежем, а столбец называется атрибутом.

В концептуальной модели реляционной БД аналогом таблицы является сущность (entity), с определенным набором свойств - атрибутов, способных принимать определенные значения (набор допустимых значений - домен).

Ключевой элемент таблицы (ключ, regular key) - такое ее поле (простой ключ) или строковое выражение, образованное из значений нескольких полей (составной ключ), по которому можно определить значения других полей для одной или нескольких записей таблицы. На практике для использования ключей создаются индексы - служебная информация, содержащая упорядоченные сведения о ключевых значениях. В реляционной теории и концептуальной модели понятие "ключ" применяется для атрибутов отношения или сущности.

Первичный ключ (primary key) - главный ключевой элемент, однозначно идентифицирующий строку в таблице. Могут также существовать альтернативный (candidate key) и уникальный (unique key) ключи, служащие также для идентификации строк в таблице.

В реляционной теории первичный ключ - минимальный набор атрибутов, однозначно идентифицирующий кортеж в отношении.

В концептуальной модели первичный ключ - минимальный набор атрибутов сущности, однозначно идентифицирующий экземпляр сущности.

Связь (relation) - функциональная зависимость между объектами. В реляционных базах данных между таблицами устанавливаются связи по ключам, один из которых в главной (parent, родительской) таблице - первичный, второй - внешний ключ - во внешней (child, дочерней) таблице, как правило, первичным не является и образует связь "один ко многим" (1:N). В случае первичного внешнего ключа связь между таблицами имеет тип "один к одному" (1:1). Информация о связях сохраняется в базе данных.

Внешний ключ (foreign key) - ключевой элемент подчиненной (внешней , дочерней)

СУБД – комплекс языковых и программных средств, предназначенных для создания, ведения и совместного использования БД несколькими пользователями. СУБД позволяет: создавать БД; вставлять, обновлять, удалять и извлекать информацию из БД; предоставляет контролируемый доступ к базе данных. Взаимосвязь основных терминов в области проектирования баз данных и работы с ними.

# 2 ЗАДАЧИ И НАЗНАЧЕНИЕ ПО

Разработка приложения в среде Microsoft Visual Studio.NET (для профессионалов). Программный код должно написан на языке C# под библиотекой ASP.NET. Работа пользователя с приложением построена на основе диалогового интерфейса.

Разработать автоматизированную систему расчета за услуги, предоставляемые организациями «Гостиничный бизнес: учет предоставления дополнительных услуг (бар, пляж, салон красоты, дискотека). Сформировать отчет об использовании дополнительных услуг в зависимости от времени года и типа номера».

**Задачи проекта:**

Разработать автоматизированную систему расчета за услуги, предоставляемые организациями, которая автоматизирует процессы регистрации своих клиентов, учета использования услуг, выполнения расчетов стоимости предоставленных каждому пользователю услуг, сбора статистических данных.

Для задачи необходимо разработать подсистему *учета предоставления дополнительных услуг*, выполняющую следующие функции:

* регистрация и изменение новых пользователей (только персоналы гостиницы);
* регистрация и изменение новых посетителей (клиенты);
* установка и отключение оконечных услуг и заказов;
* регистрация изменений тарифных планов услуг и комнат;
* статистическая информация (учет предоставления дополнительных услуг (бар, пляж, салон красоты, дискотека) и формирование отчетов в зависимости от времени года и типа номера» для гостиницы).

Создание базы данных предусматривает создание базовых объектов реляционных БД современной архитектуры на концептуальном, логическом и физическом уровне представления.

Автоматизация обработки данных БД предполагает:

* использование встроенных в СУБД инструментальных средств для автоматизации работы с данными объектов БД;
* использование структурированного языка запросов SQL для манипулирования данными;
* использование модели ADO.NET — набора библиотек, поставляемый с Microsoft .NET Framework 4.0 и предназначенный для взаимодействия с различными хранилищами данных из .NET — приложений.

Приложение «Подсистема учета предоставления дополнительных услуг (бар, пляж, салон красоты, дискотека)» предназначено для операторов и администраторов, принимающие заказов гостиничных услуг.

# 3 ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

## 3.1 Техническое обеспечение

Персональные компьютеры управления дополнительных услуг функционируют в рамках локальной вычислительной сети, которая представляет собой единый комплекс вычислительных машин, связанных с сервером и между собой с помощью кабельной системы и активного сетевого оборудования. Сетевые ресурсы доступны пользователям в соответствии с правами доступа.

Вычислительная сеть, функционирующая в управлении, имеет типовую конфигурацию горизонтальной системы на основе медного кабеля UTP категории 5е и Ethernet-коммуникатора с пропускной способностью 100 Мб/с. Рабочие места пользователей и места установки сетевых принтеров оснащены розетками типа RJ-45.

## 3.2 Компьютер пользователя

* операционная система: Windows XP, …/SQLEXPRESS (рекомендуется MS Windows XP/Server 2008 и выше);
* процессор Intel Pentium II 400 МГц и выше (рекомендуется Intel Pentium III 866 МГц и выше);
* оперативная память 128 Мбайт и выше (рекомендуется 256 Мбайт);
* жесткий диск (при установке используется около 120 Мбайт);
* USB-порт;
* SVGA дисплей.

## 3.3 Информационное обеспечение

Эффективность работы управления в значительной степени зависит от информационной базы. Информационная база представляет собой совокупность данных о законодательных актах, регламентирующих деятельность управления, и изменения в них.

Информация способствует реализации всех функций управления:

* плановой;
* организационной;
* контрольной;
* регулирующей;
* распределительной.

Основной источник первичной информации – это нормативно правовые акты и местного значения, на основании которых был разработан план мероприятий по ведению реестра объектов потребительского рынка: Законы Украины, уставы и положения местного значения и др.

Другой источник информации – вся внутренняя документация, которая появляется в ходе работы непосредственно с реестром: различного вида письма, план работы отдела на месяц, еженедельный регламент работы отдела, поручения от начальника управления и др.

автоматизация реестр обслуживание экономический

## 3.4 Программное обеспечение

На всех компьютерах управления, в том числе и в отделе лицензирования и реестра, установлена операционная система Microsoft Windows XP и стандартный пакет программ: Microsoft Office, 7Zip, Adobe Reader, так же активно используется электронная почта.

Организовать совместную и эффективную работу всех специалистов управления, задействованных в решении задач, в рамках единого информационного пространства возможно лишь используя современные средства автоматизации.

# 4 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Техническое задание (ТЗ) – исходный документ на проектирование технического объекта (изделия). ТЗ устанавливает основное назначение разрабатываемого объекта, его технические характеристики, показатели качества и технико-экономические требования, предписание по выполнению необходимых стадий создания документации (конструкторской, технологической, программной и т.д.) и её состав, а также специальные требования.

Все изменения, дополнения и уточнения формулировок ТЗ обязательно согласуются с заказчиком и им утверждаются. Это необходимо и потому, что в случае обнаружения в процессе решения проектной задачи неточностей или ошибочности исходных данных возникает необходимость определения степени вины каждой из сторон-участниц разработки, распределения понесенных в связи с этим убытков.

В процессе проектирования было создано и утверждено техническое задание на разработку корпоративной системы учета предоставления дополнительных систем, которое приведено ниже.

**Содержание**

4.1. Введение

4.1.1. Наименование программы

4.1.2. Назначение и область применения

4.2. Требования к программе

4.2.1. Требования к функциональным характеристикам

4.2.2. Требования к надежности

4.2.2.1. Требования к обеспечению надежного функционирования программы

4.2.2.2. Время восстановления после отказа

4.2.2.3. Отказы из-за некорректных действий пользователей системы

4.3. Условия эксплуатации

4.3.1. Требования к квалификации и численности персонала

4.3.2. Требования к составу и параметрам технических средств

4.3.3. Требования к информационной и программной совместимости

4.3.3.1. Требования к информационным структурам и методам решения

4.3.3.1.1. Структура баз данных

4.3.3.1.2. Требования к запросам пользователей данных из базы

4.3.3.2. Требования к исходным кодам и языкам программирования

4.3.3.3. Требования к программным средствам, используемым программой

4.3.3.4. Требования к защите информации и программ

4.3.4. Специальные требования

4.4. Требования к программной документации

4.4.1. Предварительный состав программной документации

4.5. Стадии и этапы разработки

4.5.1. Стадии разработки

4.5.2. Этапы разработки

4.5.3. Содержание работ по этапам

4.6. Порядок контроля и приемки

4.6.1. Виды испытаний

4.6.2. Общие требования к приемке работы

**4.1. Введение**

**4.1.1. Наименование программы**

Наименование программы: "Корпоративаня подсистма «Гостиница» учета предоставления дополнительных услуг".

**4.1.2. Назначение и область применения**

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

- единый учет заказов (свободен или занят номеров, тип: эконом- класс, средний, люкс);

- единый учет дополнительных услуг (салон красоты, дискотека, пляж, сауна);

- поддержка учета посетителей и нагрузок;

- формирование отчетов;

- детализированный расчет стоимости конкретной услуги.

Программа представляет собой клиент-серверное приложение работающее под ОС Windows.

**4.2. Требования к программе**

**4.2.1. Требования к функциональным характеристикам**

Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

4.2.1.1. Разделение пользователей на группы:

4.2.1.1.1. Администратор;

4.2.1.1.2. Операторы;

4.2.1.2. Возможность авторизации в информационной системе;

4.2.1.3. Возможность добавление, редактирование личных данных клиентов;

4.2.1.3. Возможность расчёта дополнительных услугов;

4.2.1.4. Возможность вывода отчёта за определённый период;

4.2.1.5. Возможность добавления новых пользователей (клиентов/посетителей) в систему;

**4.2.2. Требования к надежности**

**4.2.2.1. Требования к обеспечению надежного функционирования программы**

Надежное (устойчивое) функционирование программы должно быть обеспечено выполнением Заказчиком совокупности организационно-технических мероприятий, перечень которых приведен ниже:

а) организацией бесперебойного питания технических средств;

б) использованием лицензионного программного обеспечения;

в) регулярным выполнением требований ГОСТ 51188-98. Защита информации. Испытания программных средств на наличие компьютерных вирусов

**4.2.2.2. Время восстановления после отказа**

Время восстановления после отказа, вызванного сбоем электропитания технических средств (иными внешними факторами), не фатальным сбоем (не крахом) операционной системы, не должно превышать 30-ти минут при условии соблюдения условий эксплуатации технических и программных средств.

Время восстановления после отказа, вызванного неисправностью технических средств, фатальным сбоем (крахом) операционной системы, не должно превышать времени, требуемого на устранение неисправностей технических средств и переустановки программных средств.

**4.2.2.3. Отказы из-за некорректных действий пользователей системы**

Отказы программы вследствие некорректных действий пользователя при взаимодействии с информационной системы недопустимы.

**4.3. Условия эксплуатации**

**4.3.1. Требования к квалификации и численности персонала**

Минимальное количество персонала, требуемого для работы программы, должно составлять не более 2 штатных единиц — администратор и оператор гостиницы. Администратор должен иметь высшее профильное образование и сертификаты компании-производителя операционной системы. В перечень задач, выполняемых администратором, должны входить:

а) задача поддержания работоспособности технических средств;

б) задачи установки (инсталляции) и поддержания работоспособности системных программных средств — операционной системы;

в) задача установки (инсталляции) программы.

г) задача создания резервных копий базы данных.

**4.3.2. Требования к составу и параметрам технических средств**

4.3.3.1. В состав технических средств должен входить персональный компьютер (ПК), включающий в себя:

4.3.3.1.1. процессор Pentium-IV 2GHz, не менее;

4.3.3.1.2. оперативную память объемом, 512Мегабайт, не менее;

4.3.3.1.3. HDD, 20 Гигабайт, не менее;

4.3.3.1.4. операционную систему Windows XP и выше;

**4.3.3. Требования к информационной и программной совместимости**

**4.3.3.1. Требования к информационным структурам и методам решения**

База данных работает под управлением ..//SQLEXSPRESS

**4.3.3.1.1. Структура баз данных**



**4.3.3.1.2. Требования к запросам пользователей данных из базы**

Операторы и администраторы работают с базой данных через подсистемы информационной системы.

**4.3.3.2. Требования к исходным кодам и языкам программирования**

Дополнительные требования не предъявляются.

**4.3.3.3. Требования к программным средствам, используемым программой**

Системные программные средства, используемые программой, должны быть представлены лицензионной локализованной версией операционной системы Windows XP и выше.

**4.3.3.4. Требования к защите информации и программ**

Требования к защите информации и программ не предъявляются.

**4.3.4. Специальные требования**

Программа должна обеспечивать одновременную работу пользователей посредством клиент-серверной архитектуры.

**4.4. Требования к программной документации**

**4.4.1. Предварительный состав программной документации**

Состав программной документации должен включать в себя:

**4.4.1.1. техническое задание;**

**4.4.1.2. программу и методики испытаний;**

**4.4.**1.3. руководство оператора;

**4.5. Стадии и этапы разработки**

**4.5.1. Стадии разработки**

Разработка должна быть проведена в три стадии:

1. разработка технического задания;

2. рабочее проектирование;

3. внедрение.

**4.5.2. Этапы разработки**

На стадии разработки технического задания должен быть выполнен этап разработки, согласования и утверждения настоящего технического задания.

На стадии рабочего проектирования должны быть выполнены перечисленные ниже этапы работ:

1. разработка программы;

2. разработка программной документации;

3. испытания программы.

На стадии внедрения должен быть выполнен этап разработки подготовка и передача программы.

**4.5.3. Содержание работ по этапам**

На этапе разработки технического задания должны быть выполнены перечисленные ниже работы:

1. постановка задачи;

2. определение и уточнение требований к техническим средствам;

3. определение требований к программе;

4. определение стадий, этапов и сроков разработки программы и документации на неё;

5. согласование и утверждение технического задания.

На этапе разработки программы должна быть выполнена работа по программированию (кодированию) и отладке программы.   
На этапе разработки программной документации должна быть выполнена разработка программных документов в соответствии с требованиями к составу документации.

На этапе испытаний программы должны быть выполнены перечисленные ниже виды работ:

1. разработка, согласование и утверждение и методики испытаний;

2. проведение приемо-сдаточных испытаний;

3. корректировка программы и программной документации по результатам испытаний. На этапе подготовки и передачи программы должна быть выполнена работа по подготовке и передаче программы и программной документации в эксплуатацию на объектах Заказчика.

# 5 АНАЛИЗ ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ ПОДСИСТЕМЫ УЧЕТА ПРЕДОСТАВЛЕНИЯ ДОПОЛНИТЕЛЬНЫХ УСЛУГ

## 5.1 Требования заказчика

### 5.1.1 Мандатные требования

5.1.1.1 Разработать подсистему учета предоставления дополнительных услуг;

5.1.1.2 Система должна быть разработана под платформу Windows

5.1.1.3 У системы должны быть уровни доступа:

- оператор;

- администратор.

5.1.1.4 Оператору должна быть доступна возможность авторизации в системе.

5.1.1.4.1 Форма регистрации должна содержит поля ввода:

- Имя пользователя (Кличка);

- ФИО;

- должность;

- пароль;

- повторный пароль.

5.1.1.4.2 Ввод оператором персональной информации при авторизации:

- логин;

- пароль.

5.1.1.4.3 Анализ введенных оператором данных.

5.1.1.4.4 Вывод сообщений при некорректной обработке запроса на вход в систему.

5.1.1.5 У оператора должна быть возможность просмотра информации:

- список текущих заказов;

- форма тарификации;

- отчеты.

5.1.1.6 Оператору должна предоставляться возможность работы с клиентами (или посетителями):

5.1.1.6.1 Формировать заказ с дополнительными услугами.

5.1.1.6.2 Выдача чека.

5.1.1.6.3 Оператор заполняет заказ.

5.1.1.6.4 Оператор должен обработать заказ:

- проверить заказ;

- проверить свободных комнат;

- удалить заказ;

- распечатать чек;

- распечатать отчет.

5.1.1.7 Оператор должен формировать отчет по заказам за день, месяца, квартал, год и выводить на экран.

5.1.1.8 Разработчик должен предоставить сопровождение системы в течение года.

5.1.1.9 Ввод информации осуществляется оператор и администратор.

5.1.1.10 Ввод данных в БД осуществляется только администратор.

### 5.1.2 Ограничительные требования

5.1.2.1 Разработчик должен предоставлять и поддерживать:

- очередную версию системы;

- исправление ошибок и багов;

- создание и обновление системы и также БД;

- очередною версию приложения, если в объектах базы данных были изменения;

5.1.2.2 Система должно корректно обрабатывать ошибки (исключения) и выдавать сообщения о непредвиденной ситуации.

5.1.2.3 Список заказов и клиентов должен храниться в базе данных;

5.1.2.4 Все текстовые сообщения в системе должны быть на русском языках.

5.1.2.5 Русскоязычный интерфейс.

### 5.1.3 Требования ПО

#### 5.1.3.1 Функциональные требования

5.1.3.1.1 Управление доступом:

- администратор;

- оператор.

5.1.3.1.2 Управление экранными формами.

5.1.3.1.2.1 Режим оператора.

5.1.3.1.2.1.1 Ввод информации для формирования заказов:

а) заполнение личных данных клиентов:

- ФИО;

- дата рождения;

- пол;

- идентификационный код;

- паспорт (номер и серия, кем и дата выдачи в одном поле);

- адрес проживания;

- телефон.

б) заполнение заказа:

- ФИО клиента (вместе кода клиента с ФИО);

- тип комнаты:

- люкс;

- стандарт;

- эконом-класс.

- количество мест в комнате:

- одноместная;

- двухместная;

- трехместная.

- номер комнаты (система должна автоматически выдать список свободной номера);

- количество человек (от 1 до 6 человека в одной комнате);

- дата заселения;

- дата выселения;

- цена (система должна автоматически вычисления цены);

- оплата.

в) заполнение дополнительных услуг:

- услуга:

- бар;

- салон красоты;

- пляж;

- дискотека.

- дата использования услуги;

- количество человека;

- оплата.

5.1.3.1.2.1.2 Просмотр заказов.

5.1.3.1.2.1.4 Просмотр дополнительных услуг в выбранном заказе.

5.1.3.1.2.1.5 Анализ и обработка данных:

- подтверждение заказа;

- подтверждение оплаты;

- удаление заказов;

- удаление дополнительных услуг в выбранном заказе.

- добавление новых клиентов;

- удаление клиентов.

5.1.3.1.2.2 Режим администратора.

5.1.3.1.2.2.1 Инсталляция системы и подключить с сервером БД Server SQLExpress.

5.1.3.1.2.2.2 Инсталляция БД.

5.1.3.1.2.2.3 Добавление, редактирование, удаление данных в БД, также обработка формирования отчетов.

#### 5.1.3.2 Нефункциональные требования

5.1.3.2.1. Требования к ПК:

- процессор с тактовой частотой не ниже 2 ГГц;

- жесткий диск не ниже 100 ГБ;

- оперативная память (ОЗУ) не ниже 512 МБ;

- видеопамять не ниже 256 МБ.

5.1.3.2.2 Требования к уровню подготовки оператора – квалификация оператора и знание пользования ПК и браузеров.

5.1.3.2.3 Система требуется наличие доступа к сети интернета со скоростью не менее 1 Мбит/сек.

5.1.3.2.4 Система установлена ОС Windoiws

5.1.3.2.5 Требуется наличие одного из перечисленных браузеров:

- chrome;

- mozilla firefox;

- opera;

- safari;

- IE (браузер Windows).

### 5.1.4 Построение диаграммы вариантов использования

Диаграмма прецедентов (диаграмма вариантов использования) в UML – диаграмма, отражающая отношения между актёрами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне [3].

Прецедент – возможность моделируемой системы (часть её функциональности), благодаря которой пользователь может получить конкретный, измеримый и нужный ему результат. Прецедент соответствует отдельному сервису системы, определяет один из вариантов её использования и описывает типичный способ взаимодействия пользователя с системой. Варианты использования обычно применяются для спецификации внешних требований к системе [4].

Актёр («эктор») – стилизованный человечек, обозначающий набор ролей пользователя (понимается в широком смысле: человек, внешняя сущность, класс, другая система), взаимодействующего с некоторой сущностью (системой, подсистемой, классом). Актёры не могут быть связаны друг с другом (за исключением отношений обобщения/наследования) [5].

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик.

Выполнив анализ предметной области, была составлена диаграмма вариантов использования системы «Гостиницы», изображена на рисунке 5.1.

В диаграмме вариантов использования используется сценарий взаимодействия между "Оператор" и "Администратором".

В ходе анализа для данного сценария было выделено 2 действующих лица: "Оператор" и "Администратор". Для каждого из них были выделены прецеденты.

Полученная диаграмма вариантов использования АСР "Учет дополнительных услуг" показана на рисунке 5.1.

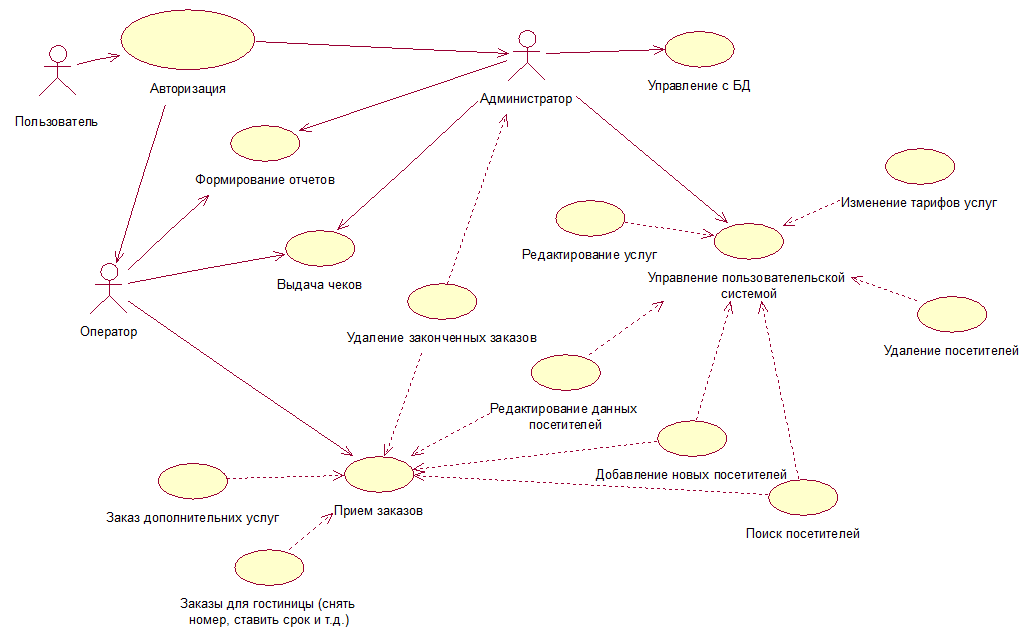


Рисунок 5.1 – Диаграмма вариантов использования системы

# 6 КОНЦЕПТУАЛЬНЫЕ ПРОЕКТИРОВАНИЯ

## 6.1 Характеристика предприятия и анализ существующей АИР системы

Гостиница занимается предоставлением услуг в сфере гостиничного бизнеса.

Данная система создается для гостиницы, в которой может выдать номер для прибывшего клиента. В зависимости от выбора клиента его могут заселить в один из номеров: одноместный, двуместный, трехместный, либо тип комната – эконом-класс, стандарт и люкс. По решению администрации гостиницы, стоимость одних суток проживания в одноместном номере составляет 150 гривен, двуместном – 300, в номере люкс – 500 гривен. Т.е. при разработке программы необходимо учесть эти инструкции.

Гостиница относится к категории мини-отелей, поскольку предоставляет для размещения гостей лишь 27 номеров следующих категорий (тип комнаты и место в комнате) и их цены, представленных в таблице 6.1.

Таблица 6.1 – Категории номеров и их цены

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Категория номера** | **Место** | **Количество**  **номера** | **Номера комнаты** | **Стоимость (грн в день)** |
| Эконом-класс | одноместная | 3 | 111, 112, 113 | 150 |
| двуместная | 3 | 121, 122, 123 |
| трехместная | 3 | 131, 132, 133 |
| Стандарт | одноместная | 3 | 211, 212, 213 | 300 |
| двуместная | 3 | 221, 222, 223 |
| трехместная | 3 | 231, 232, 233 |
| Люкс | одноместная | 3 | 311, 312, 313 | 500 |
| двуместная | 3 | 321, 322, 323 |
| трехместная | 3 | 331, 332, 333 |

В гостинице гостям предоставляются помимо основных услуг: проживание и ряд дополнительных услуг: бар, дискотека, пляж и салоны красоты. В таблице 6.2 устанавливает цены на дополнительные услуги.

Таблица 6.2 – цены на дополнительные услуги

|  |  |
| --- | --- |
| **Типы услуги** | **Цена (грн)** |
| Бар | 300 |
| Дискотеки | 50 |
| Пляж | 150 |
| Салон красоты | 120 |

В день выезда гостя из отеля, автоматически производится расчет стоимости проживания и дополнительных услуг. Установить количество человека в одной комнате или услуг и общую сумму по проживанию и по дополнительным услугам рассчитывается автоматически. С гостя взимается оплата и отбивается чек. После чего выдаются отчетные документы: счет-фактура, акт выполненных работ, которые создаются автоматически.

Вся информация о клиента и о проживании гостя и его дополнительные услуги храниться в БД системы, сконфигурированной в сервере SQLExpress. Все расчеты и формирование отчетных документов производятся автоматически.

В гостиницах используют два вида расчетов – наличный и безналичный (перевод денег на счет гостиницы). В отелях высокого класса в качестве безналичного расчета принимаются кредитные карты. Ими расплачиваются индивидуальные клиенты. Для расчетов с помощью пластиковых карт должно быть специальное оборудования: импринтер или электронный терминал.

Оплата за проживание производится в соответствии единым расчетным часом - 12 часов текущих суток по местному времени. Расчет оплаты за проживание начинают считать с 12 часов той даты, когда гость заехал, независимо от фактического часа заезда.

При проживании менее суток оплата взимается за целые сутки независимо от расчетного часа.

При задержке выезда не более, чем на 6 часов после расчетного часа (с 12:00 до 17:59) взимается почасовая оплата, при задержке выезда в период с 18:00 до 23:59 взимается оплата за половину суток. При задержке выезда более, чем на 12 часов, оплата взимается за сутки.

Если расчет оплаты делается вручную, то гостю выписывается счет. Счет оформляется в трех экземплярах, каждый из которых имеет один и тот же регистрационный номер. Первый экземпляр счета со штампом гостиницы «оплачено» выдается гостю. Второй экземпляр в конце смены передается в бухгалтерию. Третий остается в книжке счетов. Для их хранения в службе существует специальная картотека счетов, в которой счета хранятся в специальном кармашке по датам выезда.

Если расчет производится с помощью автоматического учета мест, то счет выдается системой.

## 6.2 Функциональная модель IDEF0

Описание бизнес процессов сайта является очень важным шагом в процессе автоматизации его деятельности. В процессе описания бизнес-процессов происходит формализация всей деятельности, производимой подсистемой учета дополнительных услуг, выделяются ключевые моменты автоматизации.

В настоящее время очевиден факт, что успешное функционирование человеко-машинных информационных систем и технологий определяется качеством проектирования. Проектирование имеет целью обеспечить эффективное функционирование подсистемы. Именно качественное проектирование обеспечивает создание такой системы, которая способна функционировать при постоянном совершенствовании её технических, программных и информационных составляющих, т.е. её технологической основы, и расширять спектр реализуемых управленческих функций и объектов взаимодействия.

Проектирование информационной системы "Приемы заказов" начинается с рассмотрения бизнес – процессов.

Для описания бизнес процессов подлежащих учету в информационной системе "Прием заказов" используется функциональное моделирование.

Функциональная модель представляет любой процесс как совокупность функциональных блоков соединенных интерфейсными дугами. Функциональный блок - действие, выраженное глагольным оборотом, интерфейсная дуга - предмет, описанный существительным с уточняющей информацией. В зависимости от места соединения дуги с блоком различают входы (слева), выходы (справа), управление (сверху), механизмы или ресурсы (снизу).

Функциональная модель предназначена для описания существующих бизнес – процессов на предприятии (так называемая модель AS-IS) и идеального положения вещей – того, к чему нужно стремиться (модель ТО-ВЕ). Методология IDEF0 предписывает построение иерархической системы диаграмм – единичных описаний фрагментов системы.

Построение модели ИС начинается с описания функционирования предприятия (системы) в целом в виде контекстной диаграммы. На рисунке 6.1 представлена контекстная диаграмма ИС «Гостиница»:

Взаимодействие системы с окружающей средой описывается в терминах входа (на рисунке 6.3 это «Посетители» и «Оплата за услуги»), выхода (основной результат процесса – «Список оказанных заказов и услуг»), управления («Законодательство Украины» и «Устав гостиницы») и механизмов («Кассиры», «Администратор», «ПК» – это ресурсы, необходимые для процесса функционирования гостиницы).

«Посетители» – те, для кого гостиница работает. Они платят гостинице деньги в качестве платы за оказываемые заказов и дополнительных услуги.

«Законодательство Украины» и «Устав гостиницы» – это правила, которыми управляется процесс функционирования гостиницы, как предприятия со своими внутренними правилами, и также обязанного «жить» согласно законодательству конкретной страны.

В оказании услуг принимает участие «Кассиры» гостиницы. Чтобы предоставить номера и дополнительных услуг, в деятельности гостиницы должны участвовать «Дополнительные услуги», «Помещение» и «Материальная база» – обстановка здания, техника в номерах, инвентарь, дополнительные услуги и т.д.

Model Name: Гостиница

Definition: Модель описывает деятельность гостиницы, а именно следующие предоставляемые ею услуги:

* предоставление номеров;
* предоставление дополнительных услуг;
* администрирование ПК.

После описания контекстной диаграммы проводится функциональная декомпозиция – система разбивается на подсистемы и каждая подсистема описывается отдельно (диаграммы декомпозиции). Затем каждая подсистема разбивается на более мелкие и так далее до достижения нужной степени подробности. В результате такого разбиения, каждый фрагмент системы изображается на отдельной диаграмме декомпозиции (Рис. 6.4).

Весь процесс «Приемы заказов и дополнительных заказов» разбивается на 3:

1) «Предоставление номеров» иллюстрирует деятельность сдачи номеров с предварительной регистрацией;

2) «Предоставление дополнительных услуг» представляет собой процесс дополнения дополнительных услуг в номерах;

3) «Администрирование ПК» – это совокупность оказываемых гостиницей услуг по внесению данных в БД, взиманию платы за переговоры и ведению учета переговоров, также вычисление цен за заказы.

4) «Выдача чек» - это распечатка чек всего заказов и услуг.

Общие стрелки, перешедшие с диаграммы верхнего уровня, опишем с помощью отчета, см. в таблице 6.4.

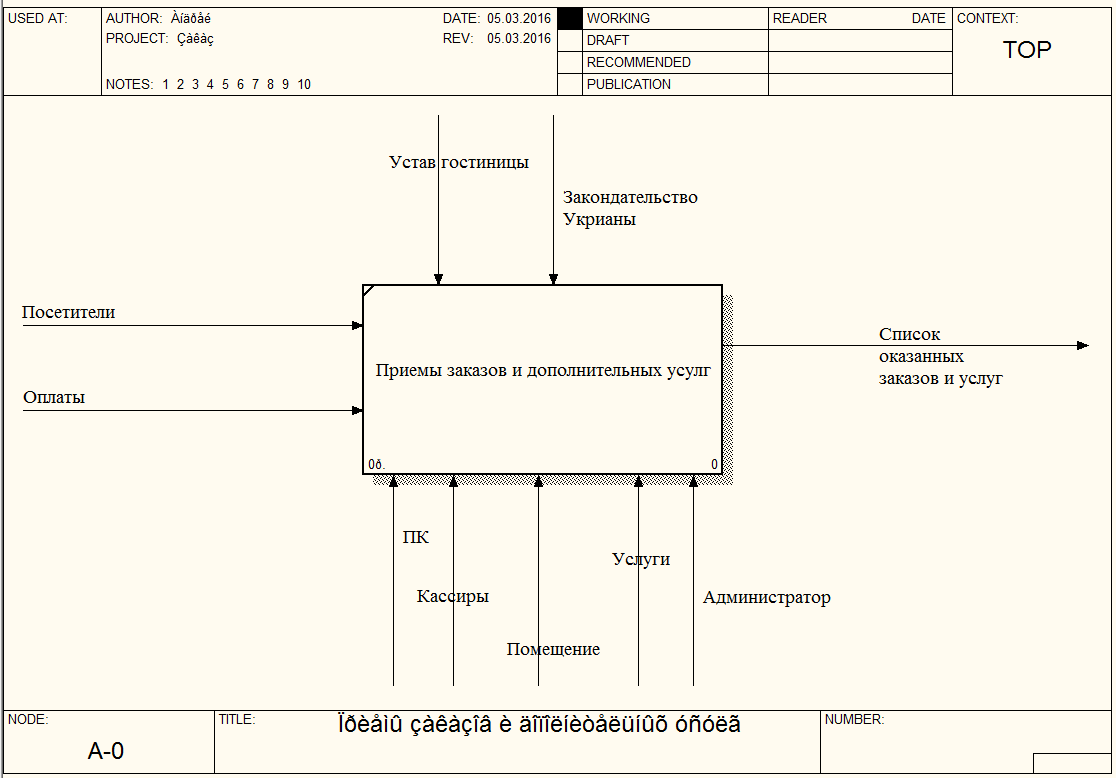


Рисунок 6.3 – Контекстная диаграмма IDEF0 «Приемы заказов и дополнительных заказов»

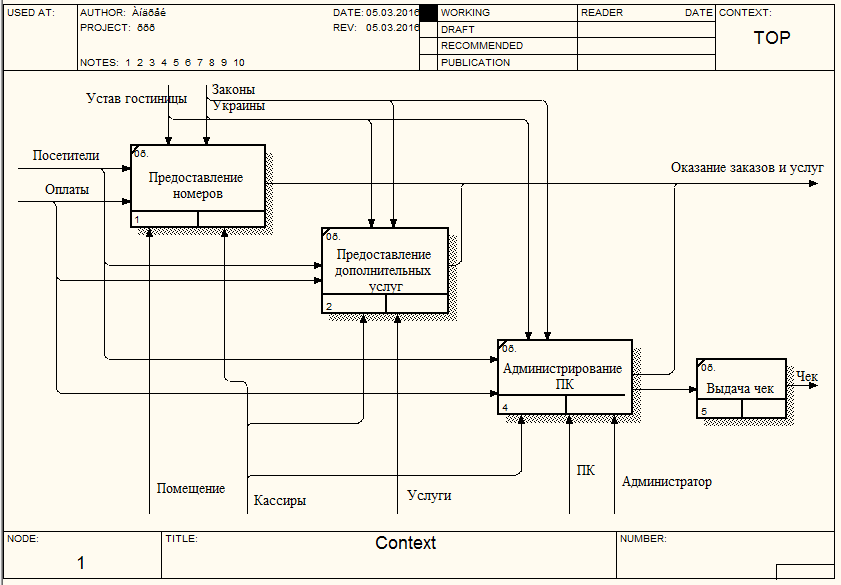


Рис. 6.4 – Диаграмма декомпозиции IDEF0 «Приемы заказов и дополнительных заказов»

Таблица 6.3 – Отчет по диаграмме IDEF0 «Приемы заказов и дополнительных заказов»

|  |  |  |
| --- | --- | --- |
| **Link Name** | **Link Definition** | **Link Status** |
| Кассиры и администратор | Люди, работающие в гостинице, осуществляющие приём клиентов, администрирование номеров, оказание дополнительных услуг. | WORKING |
| Помещение | Помещение гостиницы. Платежи по аренде этого помещения включены в оплату услуг, то есть ежемесячные расходы, покрываемые из прибыли. | WORKING |
| Плата за услуги | Часть прибыли, формирующейся из оплаты оказываемых услуг, снова возвращается в систему. | WORKING |

*Продолжение таблицы 6.1*

|  |  |  |
| --- | --- | --- |
| Посетители | Люди, создающие спрос на услуги гостиницы. | WORKING |
| Устав гостиницы | Свод правил, которым должны подчиняться все служащие гостиницы. | WORKING |
| Чек | Чек, как отчет о принятии заказов и услуг | WORKING |
| Законы Украины | Законы по защите прав потребителя, и те, которые тем или иным образом контролируют качество, оказываемых нами услуг. (Обязательство конфиденциальности почтовых пакетов, телефонных разговоров и обеспечение сохранности вещей клиентов в номерах, гарантируемое системой ключей и ответственностью персонала). | WORKING |

## 6.3 Диаграммы потоков данных (Data Flow Diagramming)

Диаграммы потоков данных (DFD) используются для описания документооборота и обработки информации. Нотация DFD включает такие понятия, как «внешняя ссылка» и «хранилище данных», что делает ее более удобной (по сравнению с IDEF0) для моделирования документооборота.

Данные, поступающие от кассиров/администраторов, хранятся на данный момент в бумажном эквиваленте. Создание клиентского приложения позволит все эти данные хранить в электронном виде и облегчит обновление данных о номерах гостиницы и постояльцах.

В отличие от стрелок IDEF0, которые представляют собой жесткие взаимосвязи, стрелки DFD показывают, как объекты (включая данные) двигаются от одной работы к другой.

На рисунке 6.5 представлена диаграмма «Диаграммы декомпозиции в нотации DFD «Администрирование ПК»», описывающая деятельность по оформлению поселения. На диаграмме представлено:

1) «Посетители» и «Кассиры», «Администратор» – это внешние ссылки, источник данных из вне модели.

2) «Устав гостиницы», «Документы клиентов» (паспорт в бумажном виде или другой удостоверяющий личность документ), «Законы Украины», «Данные о номерах гостиницы» – хранилища данных (БД).

Все работы, представленные на диаграмме выполняются «Кассиры» и «Администратор» в соответствие с «Перечнем обязанностей». Клиент запрашивает номер в гостинице («Отказ» возможен в случае отсутствия свободных номеров в гостинице). Если после «Обработки запроса» с участием «Данных о номерах» из хранилища в БД, запрос удовлетворяется: «Кассир» оформляет въезд постояльца и обновляет данные о номерах гостиницы в хранилище «Данных о номерах гостиницы».

Все это «Администратор» и «Кассир »делает, руководствуясь «правилами поселения», прописанными в «Уставе гостиницы», и «Законами и постановлениями в Украине», регламентирующими, например, обязательную идентификацию личности граждан при поселении в гостинице.

«Администратор» может управлять полным правом с БД: добавление, редактирование, удаление данных клиентов, номеров комнат, и также изменение тарифов услуг.

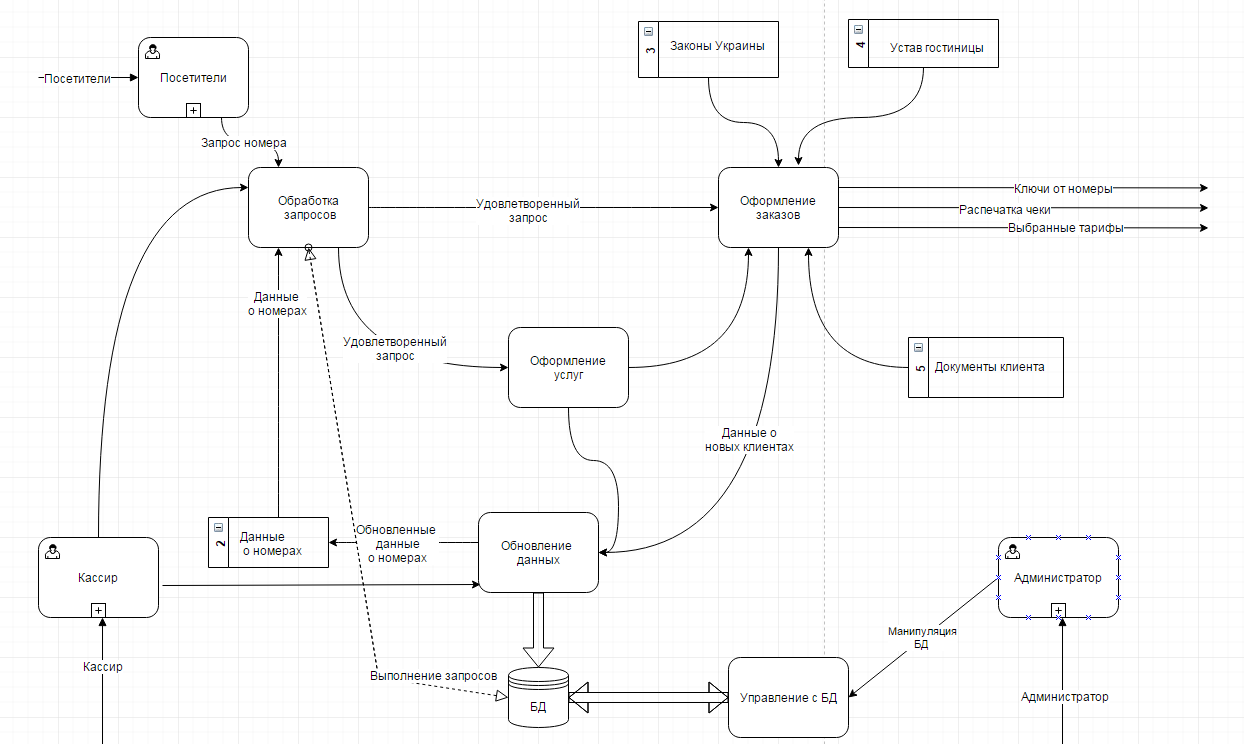


Рисунок 6.5 – Диаграммы декомпозиции в нотации DFD «Администрирование ПК»

# 7 ПРОЕКТИРОВАНИЕ БД

## 7.1 Логическое проектирование

Корпоративная система оперирует следующими основными сущностями (в скобках указаны имена этих сущностей, под которыми они реализованы в СУБД):

* клиенты (clients);
* тип номера (type\_number);
* тип комнаты (type\_rool);
* список дополнительных услуг (list\_add\_services);
* тип услуги (type\_servies);
* заказы (orders).

Для разработки БД для корпоративной подсистемы «Учет предоставления дополнительных услуг» были выделены следующие сущности:

Сущность «Клиенты», хранит необходимую информацию о клиенте или посетители, имеет такие атрибуты (в скобках указаны имена этих атрибутов, под которыми они реализованы в СУБД):

* код клиента (id\_client) – уникальный код клиента;
* ФИО клиента (FIO) – ФИО клиента;
* дата рождения (birthday) – дата рождения клиета;
* пол (pol) – пол клиента;
* идентификационный код (identif) – идентификационный код клиента;
* паспорт (passport) – паспортные данные клиента;
* адрес (address)– домашний адрес клиента;
* телефон (phone)– телефон клиента;

Сущность «Тип номера», хранит необходимую информацию о типе номера, имеет атрибуты:

* код номера (id\_number) – номер комнаты;
* имя номера (name) – имя номера (например, эконом-класс, стандарт, люкс);
* тип комнаты (col) – тип комнаты, то есть количество существующих мест.

Сущность «Тип комнаты», хранит необходимую информацию о типе комнаты, имеет атрибуты:

* код комнаты (id\_pool) – код типа комнаты;
* тип (type) – тип комнаты (например, одноместный, двухместный и т.д.);
* цена (tariff) - цена за номера.

Сущность «Список дополнительных усулг», хранит информацию о виде услуг, имеет атрибуты:

* код услуги (id\_list\_servies) – код услуги;
* код заказа (id\_orders) – код заказа;
* код услуги (id\_servies) – код услуги.

Сущность «Заказы», хранит информацию о тарифах номеров, имеет атрибуты:

* код заказа (id\_orders) – код заказа;
* дата заказа (date) – дата принятия заказа;
* начало заказа (begin) – начала посещение комнаты;
* конец заказа (end) – выселение;
* код заказа услуг (list\_servies) – код дополнительных услуг;
* оплата (payment) – оплачено или нет;
* код комнаты (id\_number) – номер комнаты.

Сущность «Тип услуги», хранит информацию о типах услуги, имеет атрибуты:

* код услуги (id\_servies) – код услуги;
* имя (name) – наименование услуги;
* цена (tariff) – цена за услуги.

Разработки базы данных для подсистемы учета дополнительных услуг гостинцы определили, что проектируемая система должна выполнять следующие действия:

* хранить информацию о клиентов, о типах номеров, о номерах комнат, об услугах, о содержаниях приказов;
* выводить список заказов;
* выводить свободные комнаты и их типы;
* выводить заказы, которые истекают срок;
* формирование годовых, сезонных, месячных отчетов.

Выше перечисленное позволяет построить полную ER-диаграмму предметной области, см. в Приложение А.

В таблице 7.1 составлено список сущностей и их назначения.

Таблица 7.1 – Назначение сущностей

|  |  |
| --- | --- |
| **Название сущности** | **Назначение** |
| clients (клиенты) | Хранение личных данных клиентов |
| type\_number (номер комнаты) | Список типов номера |
| type\_rool (тип комнаты) | Список существующих номеров |
| list\_add\_services (список дополнительных услуги) | Список дополнительных услуг заказа клиента |
| type\_services (тип услуг) | Список типов дополнительных услуг |
| Orders (заказы) | Хранение заказов |

Для реализации базы данных гостиницы была спроектирована логическая модель данных (рисунок 7.1)



Рисунок 7.1 – Логическая модель данных

## 7.2 Описание физической базы данных

Для обеспечения целостности данных хранимых в базе данных основным этапом в разработке реляционной модели является установление ограничений на вводимые данные. Все ограничения связаны с особенностями предметной области.

Для реализации БД была выбрана СУБД MySQL (см. п. 8 ообоснование выбора СУБД). На основании установленных ограничений создается физическая модель базы данных. Все сущности стали таблицами, а все их атрибуты стали полями таблицы. Каждое из полей должно иметь определенный тип. В таблицах 7.2 – 7.7 составлено поля таблиц Клиенты, Заказы, Список дополнительных услуг, Тип номера, тип комнаты имеют следующие типы:

Таблица 7.2 – Типы полей таблицы «Клиенты»

|  |  |  |  |
| --- | --- | --- | --- |
| **Название поля** | **Тип поля** | **Размер поля** | **Дополнительные** |
| Id\_client | Int | 4 | Первичный ключ |
| FIO | Varchar | 255 |  |
| Birthday | Date | Краткий формат даты |  |
| Pol | enum | «М» и «Ж» |  |
| Identif | Int | 14 |  |
| Passport | Varchar | 50 |  |
| Address | Varchar | 255 |  |
| phone | Int | 20 |  |

Таблица 7.3 – Типы полей таблицы «Заказы»

|  |  |  |  |
| --- | --- | --- | --- |
| **Название поля** | **Тип поля** | **Размер поля** | **Дополнительные** |
| Id\_ordeos | Int | 4 | Первичный ключ |
| date | Varchar | 255 |  |
| begin | Date | Краткий формат даты |  |
| end | enum | «М» и «Ж» |  |
| List\_servies | Int | 14 |  |
| payment | Boolean |  |  |
| Id\_client | Varchar | 255 | Внешний ключ |
| Id\_servies | Int | 20 | Внешний ключ |

Таблица 7.4 – Типы полей таблицы «Список дополнительных услуг»

|  |  |  |  |
| --- | --- | --- | --- |
| **Название поля** | **Тип поля** | **Размер поля** | **Дополнительные** |
| Id\_list\_servies | Int | 4 | Первичный ключ |
| Id\_orders | Int | 4 | Внешний ключ |
| Id\_servies | Int | 4 | Внешний ключ |

Таблица 7.5 – Типы полей таблицы «Тип услуг»

|  |  |  |  |
| --- | --- | --- | --- |
| **Название поля** | **Тип поля** | **Размер поля** | **Дополнительные** |
| Id\_servies | Int | 4 | Первичный ключ |
| name | Varchar | 255 |  |
| tariff | Int | 4 |  |

Таблица 7.6 – Типы полей таблицы «Тип номера»

|  |  |  |  |
| --- | --- | --- | --- |
| **Название поля** | **Тип поля** | **Размер поля** | **Дополнительные** |
| Id\_number | Int | 4 | Первичный ключ |
| name | Varchar | 255 |  |
| col | Int | 4 |  |
| Id\_tariff\_number | Int | 4 | Внешний ключ |
| Id\_pool | Int | 4 | Внешний ключ |

Таблица 7.7 – Типы полей таблицы «Тип комнаты»

|  |  |  |  |
| --- | --- | --- | --- |
| **Название поля** | **Тип поля** | **Размер поля** | **Дополнительные** |
| Id\_pool | Int | 4 | Первичный ключ |
| type | Varchar | 255 |  |
| tariff | int | 4 |  |

## 7.3 Построение физической модели БД

Для реализации поставленных задач была разработана физическая модель данных. Данная модель представлена на рисунке 7.2.



Рисунок 7.2 – Физическая модель БД

7.4 Построение запросов физической модели данных

Листинг SQL-запросов построения физической модели данных представлен в приложении Б.

# 8 ОБОСНОВАНИЕ ВЫБОРА СУБД И ИНСТРУМЕНТАЛЬНОЙ СРЕДЫ РАЗРАБОТКИ

Для разработки программного продукта была выбрана технология ASP.NET MVC 4 и язык C#, являющийся самым популярным языком для написания бизнес логики на платформе.NET, основные преимущества технологии ASP.NET MVC 4:

1) *Расширяемость*. MVC Framework также построен как ряд независимых компонентов, удовлетворяющих .NET интерфейс или построенных на абстрактном базовом классе, что позволяет легко заменять одни компоненты другими. ASP.NET MVC дизайнеры построили его таким образом, чтобы дать разработчикам три варианта выбора для каждого компонента MVC Framework:

* использовать реализацию по умолчанию компонента в его нынешнем виде (чего должно быть достаточно для большинства приложений);
* вывести подкласс реализации по умолчанию для настройки ее поведения;
* заменить компонент полностью при помощи новой реализации интерфейса или абстрактного базового класса.

2) *Жесткий контроль над HTML и HTTP.* ASP.NET MVC признает важность получения чистой, соответствующей стандартам разметки. Его встроенные методы HTML помощника предоставляют соответствующие стандартам выходные данные.

Страницы, сгенерированные ASP.NET MVC, не содержат никаких данных ViewState, поэтому они могут быть в сотни килобайт меньше, чем обычные страницы, созданные при помощи ASP.NET WebForms. Несмотря на современную широкополосную связь и быстрые подключения, эта экономия пропускной способности до сих пор чрезвычайно притягательна для конечных пользователей.

3) *Мощная система маршрутизации (роутинга)*. Стиль ссылок изменился, поскольку технология веб приложений улучшилась. Теперь можно контролировать схему ссылок и ее связь и отношение к приложению, то есть разработчик свободен в создании шаблона URL-адресов, которые являются значимыми и полезными для пользователей, без необходимости соответствовать предопределенному шаблону.

4) *Открытый исходный код.* В отличие от предыдущих платформ веб-разработки от Microsoft, можно загрузить исходный код для ASP.NET MVC и даже изменить и скомпилировать собственную версию. Это имеет неоценимое значение, когда отладка касается системы компонентов, и необходимо зайти в код (и даже прочитать комментарии программистов-создателей). Это также полезно при создании коммерческих программ для того, чтобы оценить какие возможности существуют для дальнейшего развития или как действительно работают встроенные компоненты [4].

Исходя из вышеописанных критериев, был сделан выбор в пользу разработки серверной части с использованием технологии ASP.NET MVC.

В качестве инструментальной среды разработки (IDE) была выбрана Visual Studio 2012 – это интегрированная среда разработки Microsoft (IDE), являющейся самой функциональной IDE для разработки на платформе.NET [4].

Для хранения данных был выбрана СУБД SQL Server 2008 и ORM Entitty Framework от Microsoft. Так как оба продукта от Microsoft у них есть наиболее тесная интеграция с другими продуктами [4].

# 9 РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ПОДСИСТЕМЫ УЧЕТА ПРЕДОСТАВЛЕНИЯ ДОПОЛНИТЕЛЬНЫХ УСЛУГ

## 9.1 Диаграмма классов

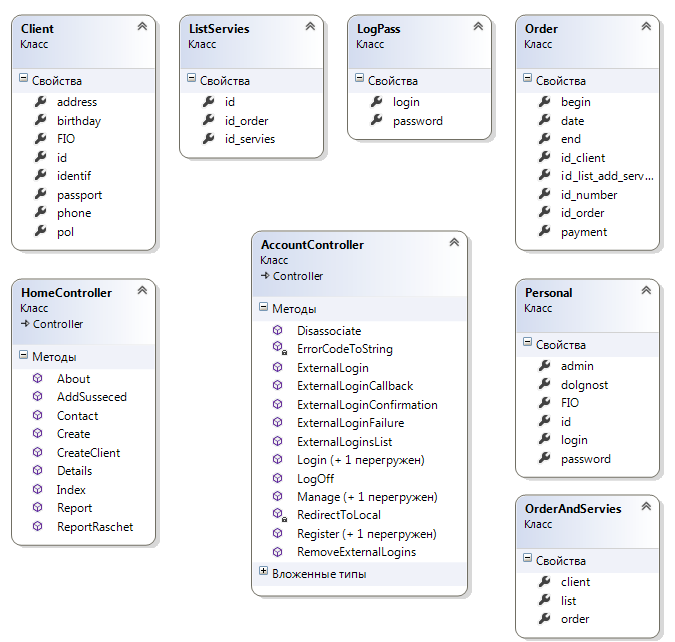


Рисунок 9.1 – Диаграмма классов системы «Гостиницы»

## 9.2 Описание диаграммы классов

Таблица 9.1 – Основные поля и методы каждого класса

|  |  |  |
| --- | --- | --- |
| **Класс Client** | | |
| **Поля** | | |
| **№** | **Поле** | **Назначение** |
| 1 | Address | адрес проживания |
| 2 | Birthday | дата дня рождения |
| 3 | FIO | ФИО |
| 4 | id | номер клиента |
| 5 | idendif | идентификационный код |

*Продолжение таблицы 9.1*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 6 | | | passport | паспорт клиента | | | |
| 7 | | | phone | телефон клиента | | | |
| 8 | | | pol | пол | | | |
| **Класс ListServies** | | | | | | | |
| **Поля** | | | | | | | |
| **№** | | | **Поле** | **Назначение** | | |
| 1 | | | id | код списка услуги | | |
| 2 | | | id\_order | код заказа | | |
| 3 | | | id\_servies | тип услуги | | |
| **Класс LogPass** | | | | | | |
| **Поля** | | | | | | |
| **№** | **Поле** | | | | **Назначение** | |
| 1 | login | | | | логин | |
| 2 | password | | | | пароль | |
| **Класс HomeController** | | | | | | |
| **Методы** | | | | | | |
| **№** | | **метод** | | | **Назначение** | | |
| 1 | | about | | | Опубликование формы автора системы | | |
| 2 | | addSusseced | | | Добавление заказов | | |
| 3 | | contact | | | Контакт с администратором | | |
| 4 | | create | | | Форма заполнения заказов | | |
| 5 | | createClient | | | Форма заполнения клиента | | |
| 6 | | report | | | Форма выбора отчета и вывод расчета заказов | | |
| 7 | | reportRaschet | | | Расчет заказов | | |
| **Класс Order** | | | | | | | |
| **Поле** | | | | | | | |
| **№** | | **Поле** | | | **Назначение** | | |
| 1 | | begin | | | начало поселения | | |
| 2 | | date | | | дата добавления заказа | | |
| 2 | | end | | | конец поселения | | |
| 3 | | id\_client | | | код клиента | | |
| 4 | | id\_list\_add\_serv | | | номер списка услуг | | |
| 4 | | id\_number | | | номер комнтаы | | |
| 5 | | id\_order | | | номер заказа | | |
| 6 | | payment | | | Оплата | | |
| **Класс AccountController** | | | | | | | |
| **Методы** | | | | | | | |
| **№** | | **Метод** | | | | **Назначение** | |
| 1 | | ErrorCodeToString | | | | Кодовая ошибка в String-виде | |
| 2 | | Login | | | | Логинирование | |
| 3 | | LogOff | | | | Отключение логина | |
| 4 | | Register | | | | Регистрация аккаунта | |
| 5 | | RemoveExternalLogins | | | | Удаление логина | |
| **Класс Personal** | | | | | | | |
| **Поля** | | | | | | | |
| **№** | | **Поле** | | | | **Назначение** | |
| 1 | | admin | | | | Право на администратора | |
| 2 | | dolgnost | | | | должность | |

*Продолжение таблицы 9.1*

|  |  |  |
| --- | --- | --- |
| 3 | id | код персонала |
| 4 | login | логин |
| 5 | password | пароль |

## 9.3 Листинги

Листинги реализации системы и подсистем исходных кодов вложены в Приложение В.

## 9.4 Скриншоты экранных работ

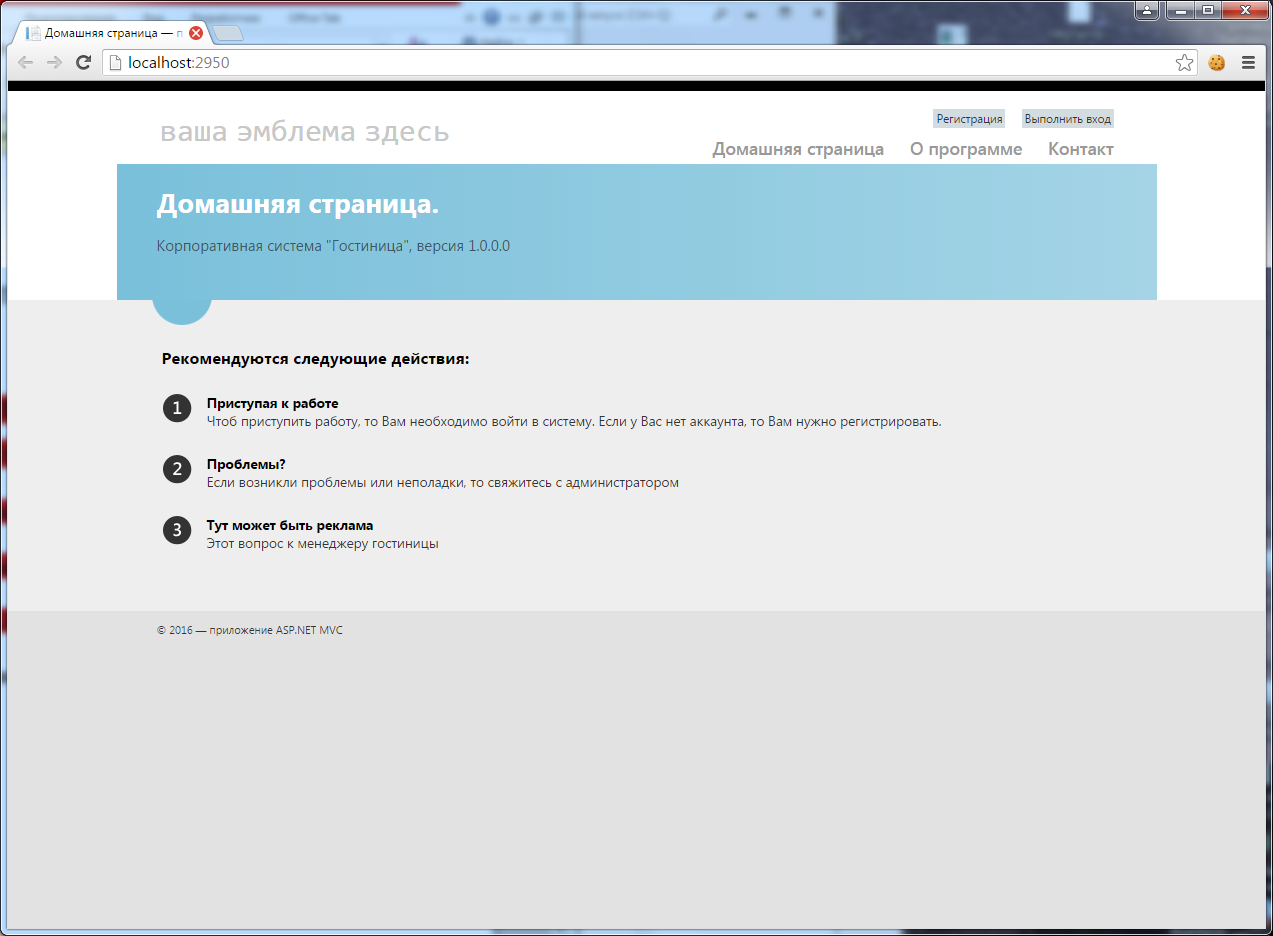


Рисунок 9.1 – Начальная страница

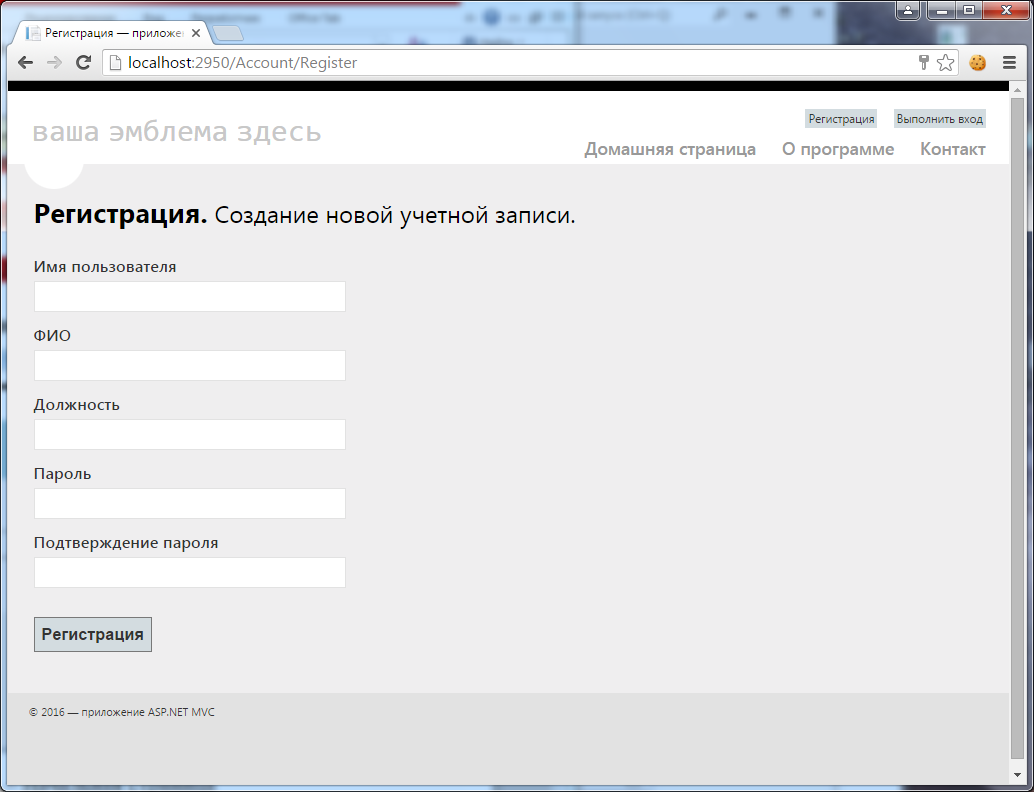


Рисунок 9.2 – Страница регистрации

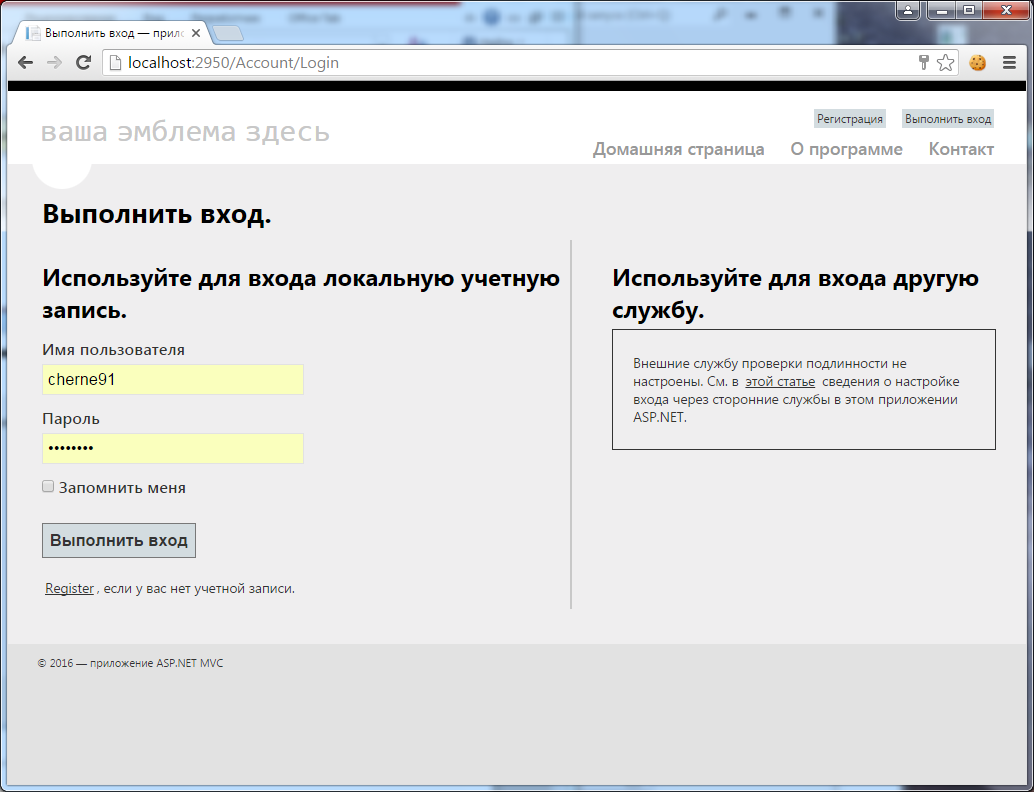


Рисунок 9.3 – Страница входа в систему

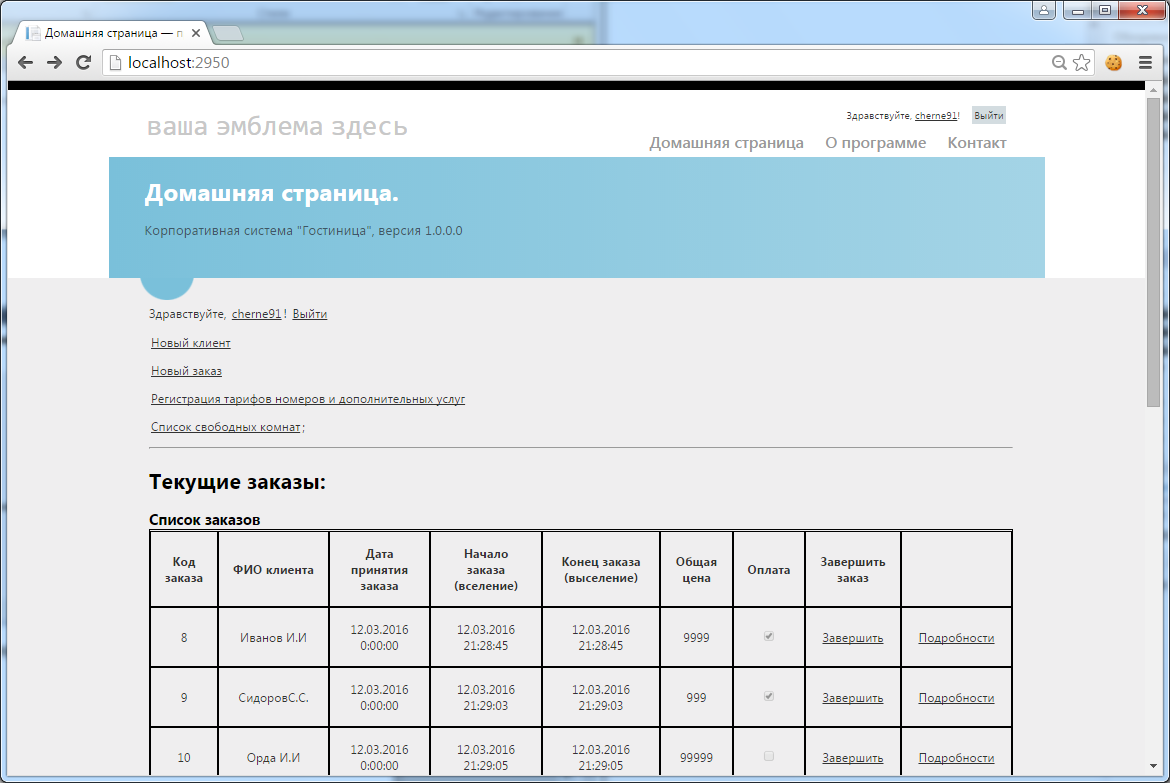


Рисунок 9.4 – Страница со списком заказов посетителей

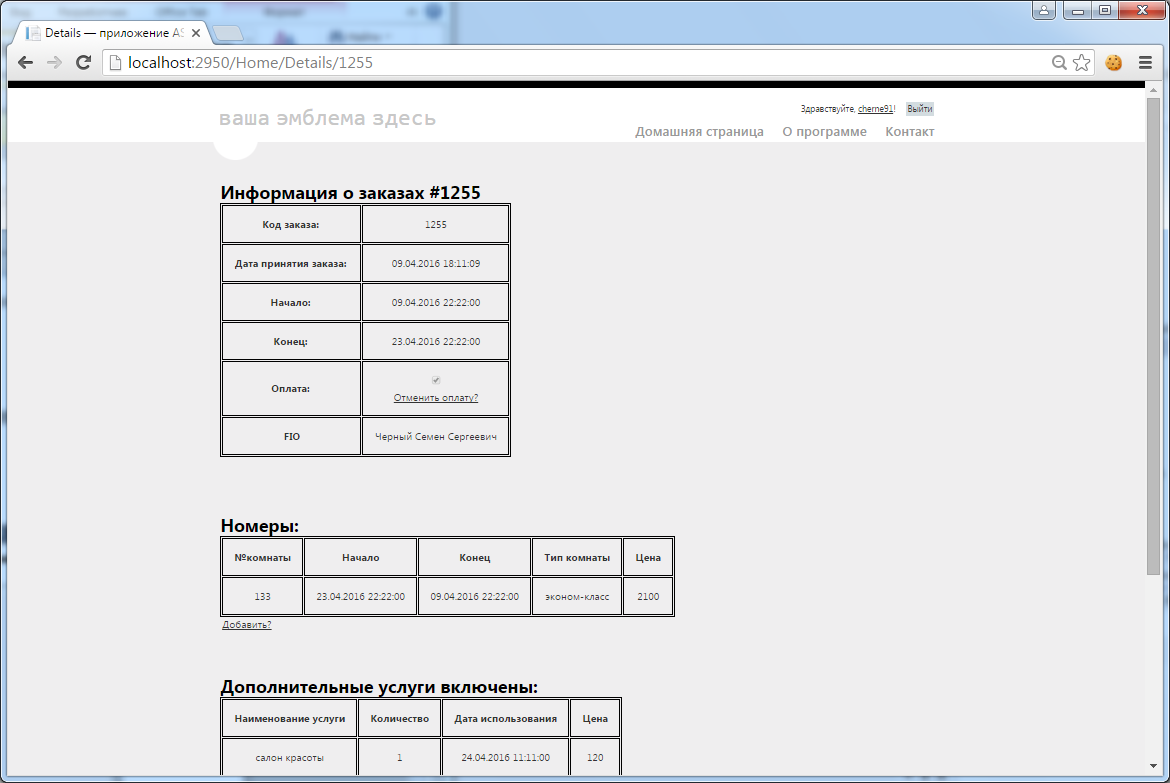


Рисунок 9.5 – Страница конкретного заказа со списком дополнительных услуг

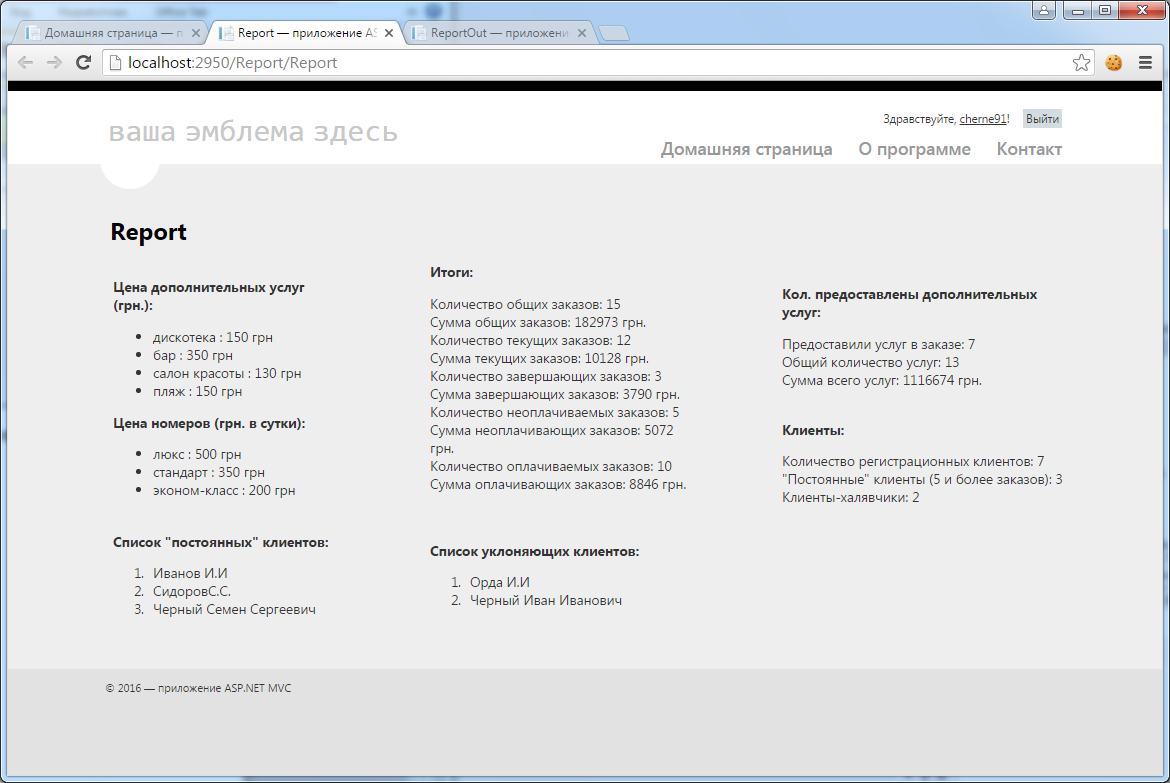


Рисунок 9.6 – Форматирование отчета – статистики

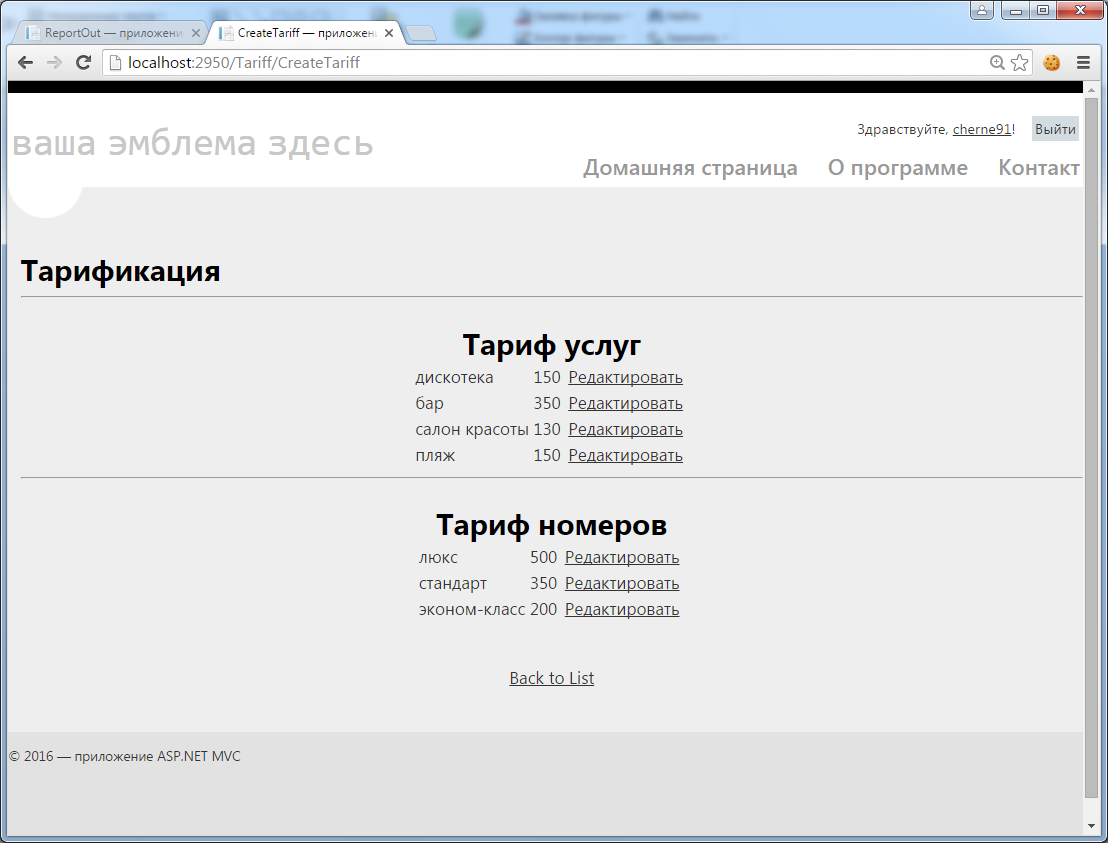
****

Рисунок 9.7 – Тарификация

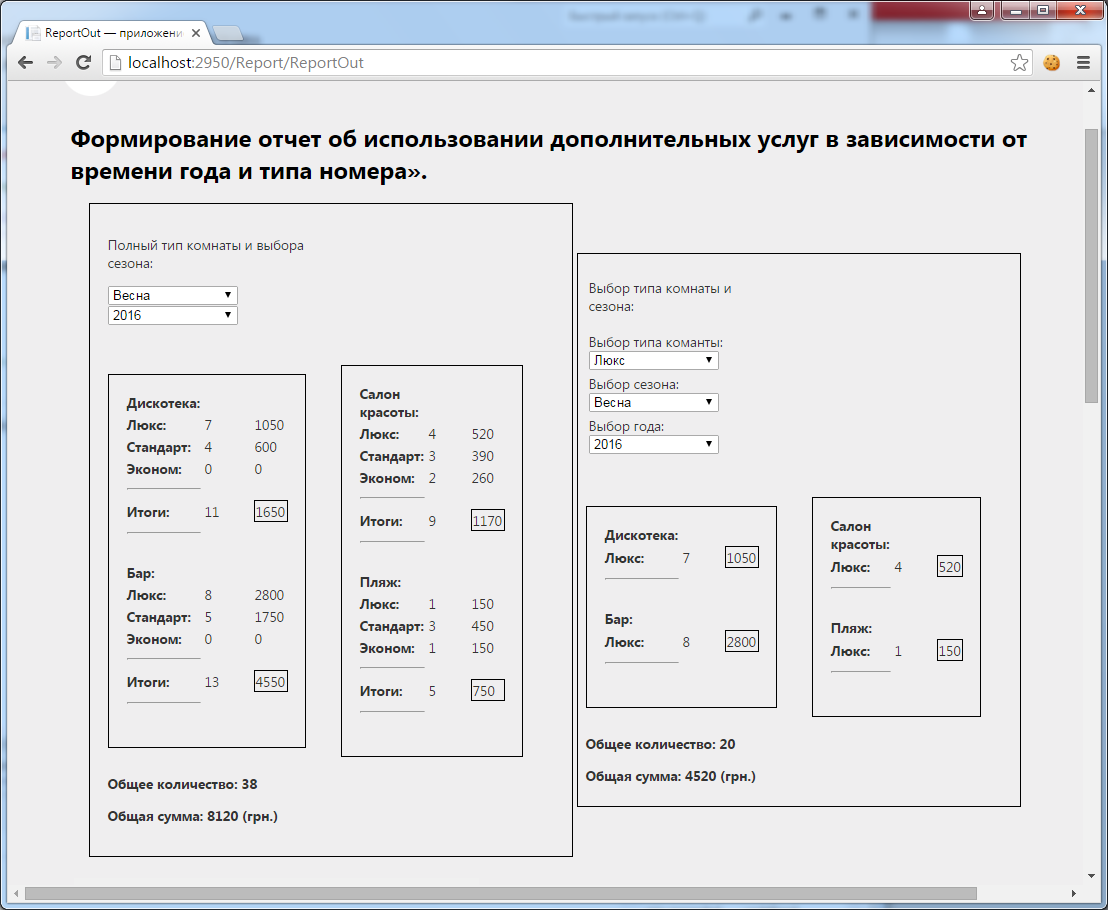
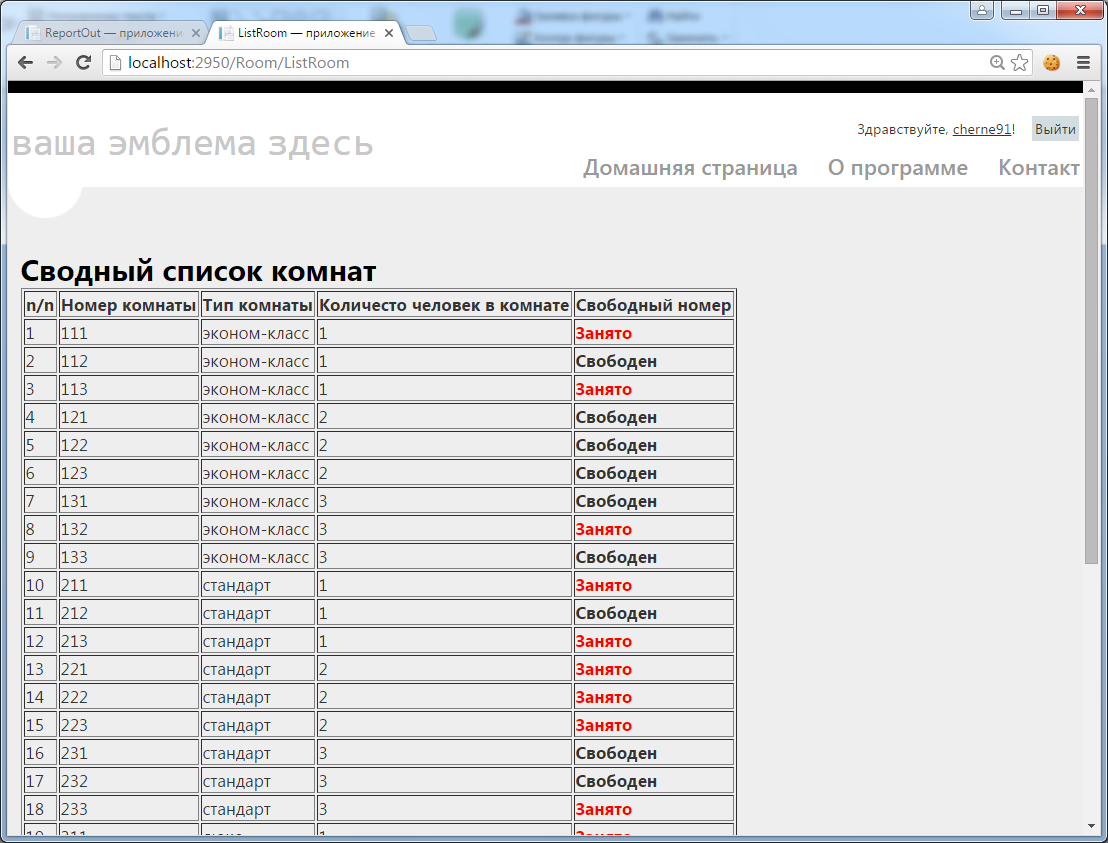
****

Рисунок 9.8 – Общая сумма всех заказов (блок слева), а блок справа – выборочная (выбор типа комнаты и сезона)



9.9 – Список свободных и занятных комнат

# 10 ТЕСТИРОВАНИЕ ПО

Тестирование подсистем, а также разработанных модулей и компонентов является одним из самых важных этапов в реализации системы «Гостиницы».

Тестирование совместной работы программных модулей называется интеграционным.

**Задачи:**

* Планирование тестирования;
* Разработка тестов;
* Формирование отладочных заданий;
* Собственно тестирование;
* Обработка результатов тестирования.

**Цели:**

* Поиск ошибок в интерфейсе;
* Поиск ошибок в связях между модулями системы;
* Устранение найденных ошибок.

**Методы:**

Метод «Белого ящика» используется в случае, когда тестировщиком является человек, который знает все процессы, происходящие в подсистеме. Как правило, в таких случаях тестировщиком является сам разработчик подсистемы. Тестирование компонентов форм на стадии разработки осуществляется в режиме отладки с использованием специального средства Visual Studio 2010 – Debug [5]. Удобство этого средства тестирования заключается в возможности пошаговой отладки создаваемого приложения в ходе выполнения программы.

Метод «Черного ящика» – метод, при котором тестировщик является человеком, не проектировавший данное ПО. Тестировщику дают тестируемое приложение и дают тестовые случаи. В ходе тестирования он должен вносить результаты тестов.

## 10.1 Планы интеграционного тестирования

Интеграционное тестирование на соответствие программного продукта требованиям со знанием внутренней структуры реализации системы.

**Идентификатор тестового плана.** Тестовый план для программного обеспечения подсистемы определения базовых признаков звуков для системы распознавания речи.

**Идентификатор** – ИТБЯ.

**Введение.** Похождение тестирования одного из подсистемы корпоративной системы «Гостиница».

**Тестируемые элементы.** Тестирование подсистемы «Учет предоставления дополнительных услуг» для системы «Гостиницы».

**Тестируемые элементы:**

* Index.cshtml
* Registrator.cshtml
* Login.cshtml
* Create.cshtml
* Details.cshtml
* Report.cshtml

В таблице приведены ниже тестируемые элементы и их соответствующие имена в исходном коде.

Таблица 10.1 – Тестируемые элементы и их соответствующие имена в исходном коде

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Тестируемые элементы** | **Вход** | **Ожидаемый результат** |
| 1 | Index.cshtml | Ввод логина и пароля | 1 Проверка login и password и возврат значения 1  2 Проверка login и password и возврат значения 0 |
| 2 | Login.cshtml | Заполнение полей логин, пароль, подтверждение пароля | 1 Переход на форму Login.cshtml  для авторизации.  2 Сообщение об ошибке о не заполнености полей |
| 3 | Registrator.cshtml | Заполнение полей для создания аккаунта оператора: ФИО, логин, пароль | 1 Переход на странице формы «регистрации» Registrator.cshtml  2 Сообщение об ошибке о не заполнености полей |
| 4 | Create.cshtml | Заполнение полей для формирования заказов: date, begin, end, payment, FIO, id\_order | 1 Проверка на заполнение полей  date, begin, end, payment, FIO, id\_order. Возврат положительного завершения создания бригады  2 Не заполнено поле  3 Ошибочно заполнено поле |

|  |  |  |  |
| --- | --- | --- | --- |
| 5 | Report.cshtml | Формирование отчета по времени: begin и end | 1 Переход на страницу Report.cshtml  2 Отображение отчетов |
| 6 | Details.cshtml | - | 1 Переход на страницу Details.cshtml  2 Отображение информации о заказе с доп. услуги |

*Продолжение таблицы 10.1*

**Подход.** Эквивалентное разбиение входных данных тестируемых функций изображено в таблице 10.2.

Таблица 10.2 – Эквивалентное разбиение входных данных

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Входное или выходное событие** | **Правильные классы эквивалентности** | **Неправильные классы эквивалентности** |
| 1 | Авторизация | Для логина – символы (буквы и цифры) с разным уровнем регистра;  Для пароля – значение равное 8 символам с разным уровнем регистра. | Символы  Кол-во символов больше или меньше 8 |
| 2 | Cоздание заказа | Заполнение всех полей | Пустое данное |
| 3 | Создание и перечисление дополнительных услуг | Правильно введенный пусть для сохранения | Пустое данное |
| 4 | Отправка заказа | Корректные заполненные данные | Некорректные данные |

**Критерии прохождения тестов.** Тест провалится, когда при сравнении фактических результатов с ожидаемыми результаты разные.

**Задачи тестирования.** Зафиксировать баг и улучшение точность работы подсистемы.

**Календарный план.** Перечислено ключевых дат и сроков представлено ниже.

Таблица 10.3 – Календарный план

|  |  |
| --- | --- |
| **Дата** | **Функции** |
| 12.03.16 – 14.03.16 | Авторизация |
| 14.03.16 – 16.03.16 | Создание заказа |
| 16.03.16 – 18.03.16 | Создание и перечисление дополнительных услуг |
| 18.03.16 – 20.03.16 | Отправка заказа |
| 20.03.16 – 21.03.16 | Подготовка отчетов |

## 10.2 Планы системного тестирования

План системных испытаний представляет собой документ, в котором обобщены планы тестирования отдельных случаев использования и предоставления инфор­мации о дополнительных видах тестирования, которые можно проводить на системном уровне.

**Идентификатор тестового плана:**СТ

**Введение.** Все тестовые мероприятия были согласованы с заказчиком. В этом разделе приведены ссылки на применяемые стандарты, а также плановые документы на программный продукт.

**Тестируемые функции.** В таблице приведены ниже тестируемые элементы и их соответствующие имена в исходном коде.

Таблица 10.4 - План системного тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Тестируемые функции** | **Входные данные** | **Ожидаемый результат** |
| СТ1 | «Регистрация» | 1. Перейти по ссылке «Регистрация»  2. Заполнить все необходимые поля в открывшейся странице  3. Нажать кнопку «Зарегистрироваться» | Зарегистрированный пользователь, записанный в БД, в соответствующие поля |
| СТ2 | «Авторизация» | 1. На главной странице нажать кнопку «Вход»  2. В окне вести логин и пароль  3. Нажать на кнопку «Вход» | Авторизованный пользователь получивший доступ к защищённой части программы. |
| СТ3 | «Создание заказа» | 1. Нажать на кнопку, создание заказа  2. Заполнение всех полей  3. Нажать на кнопку «Отправить заказ» | Созданная заявка, с заполненными полями, отправленная в БД и доступная оператору. |
| СТ4 | «Выход» | 1. Нажать на кнопку «Выход» | Разлогинивание, выход с главной формы |
| СТ5 | «Формирование отчетов» | 1. Нажать на кнопку, формирование отчетов  2. Выбрать для расчеты  3.При необходимости закрыть | Вывод отчетов |

***Подход.*** Для тестирования всех функций ПО потребуется ПК с определенными характеристиками:

* ОС: Windows 7
* CPU: AMD FX 8000 series или выше / Intel i7 Quad Core CPU
* RAM: 4 GB
* DirectX: DirectX 12

Также необходим соответственно обученный персонал с навыками оператора. Сроки тестирования функций можем увидеть в таблице 10.5

Таблица 10.5 – Календарный план

|  |  |
| --- | --- |
| **Дата** | **Функции** |
| 12.03.16 – 14.03.16 | Функция авторизации |
| 14.03.16 – 16.03.16 | Функция создания заказа |
| 16.03.16 – 18.03.16 | Функция сохранения и отправки заказа |
| 18.03.16 – 20.03.16 | Функция вывода статуса заказа |
| 20.03.16 – 21.03.16 | Подготовка отчетов |

***Критерии прохождения тестов:***которыеопределяют, про­шло ли ПО конкретный тест.

***Задачи тестирования.*** Все задачи, которые должны быть решены в ходе подготовки к тестированию и в процессе его проведения. Зависимость между этими задачами. Какова трудоёмкость работ при решении этих задач.

***Необходимый персонал.***Тестировщик Чернобривец Андрей Борисович

***Календарный план.***Перечисление ключевых дат, сроков поставки необходимых ресурсов (людей, техники, инструментальных средств и др.).

***Риск и непредвиденные обстоятельства.***

* ПО не сможет пройти утвержденные тесты.
* Критические остановки в работе ПО.
* Проблемы с сервером.

***Утверждение.*** Дегтярева Татьяна Григорьевна.

## 10.3 Результаты интеграционного тестирования

В таблице 10.6 приведен порядок выполнения тестов системных испытаний. Большая часть информации, необходимая для построения плана тестирования, будет предусмотрена планами тестирования отдельных случаев использования.

Таблица 10.6 Порядок выполнения тестов системных испытаний

|  |  |
| --- | --- |
| **Номер случая использования** | **Элемент тестирования** |
| It\_t1 | Index.cshtml |
| It\_t2 | Registrator.cshtml |
| It\_t3 | Login.cshtml |
| It\_t4 | Create.cshtml |
| It\_t5 | Details.cshtml |
| It\_t6 | Report.cshtml |

Результаты тестов приведены в таблице 10.7.

Таблица 10.7 – Результаты тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| **№ п/п** | **Описание входных данных** | **Выполняемые операции** | **Успешность выполнения** |
| 1 | Заполнение полей логин, пароль, подтверждение пароля | 1 Переход на форму для авторизации.  2 Проверка login и password и возврат значения 1.  3 Сообщение об ошибке о не заполнености полей | + |
| 2 | Ввод логина и пароля | 2 Проверка login и password и возврат значения 0 | + |
| 3 | Заполнение полей для создания клиента:  ФИО, идентификационный код, паспорт, дата рождения, адрес | 1 Проверка на заполнение полей правильными данными  2 Возврат положительного завершения создания данных клиента | + |
| 1 Проверка на заполнение полей ошибочными данными  2 Возврат отрицательного завершения создания данных клиента |
| 1 Незаполненное поле  2 Вывод ошибки о пустой строке |
| 4 | Заполнение полей для создания заказа | 1 Проверка на заполнение полей правильными данными  2 Возврат положительного завершения создания заказа | + |
| 1 Проверка на заполнение полей ошибочными данными  2 Возврат отрицательного завершения создания заказа |
| 1 Незаполненное поле  2 Вывод ошибки о пустой строке |
| 5 | Нажать на кнопку «Выход» | Разлогинивание, выход с главной формы | + |

## **10.4 Результаты выполнения системного тестирования**

В таблице 10.8 приведен порядок выполнения тестов.

Таблица 10.8 – Порядок выполнения тестов системных испытаний

|  |  |
| --- | --- |
| **Номер теста** | **Тестируемая функция** |
| СТ1 | «Регистрация» |
| СТ2 | «Авторизация» |
| СТ3 | «Создание заказа» |
| СТ4 | «Статус заказа» |
| СТ5 | «Выход» |
| СТ6 | «Формирование отчетов» |

Результаты тестирования приведены в таблице 10.9.

Таблица 10.9 – Результаты выполнения тестирования системного тестирования

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№ теста** | **Входной набор данных** | **Ожидаемый результат** | **Модуль** | **Результат тестирования** |
| СТ1 | 1. Перейти по ссылке «Регистрация»  2. Заполнить все необходимые поля в открывшейся странице  3. Нажать кнопку «Зарегистрироваться» | Зарегистрированный пользователь, записанный в БД, в соответствующие поля. | Register() | **+** |
| СТ2 | 1. На главной странице нажать кнопку «Вход»  2. В окне вести логин и пароль  3. Нажать на кнопку «Вход» | Авторизованный пользователь получивший доступ к защищённой части программы. | Login() | **+** |
| СТ3 | 1. Нажать на кнопку, «создание заказа  2. Заполнение всех полей  3. Нажать на кнопку «Отправить форму» | Созданная заявка, с заполненными полями, отправленная в БД и доступная оператору. | Index() | **+** |
| СТ4 | 1. Нажать на кнопку «Выход» | Разлогинивание, выход с главной формы | Logout() | **+** |
| СТ5 | Нажать на кнопку, формирование отчетов | Вывод отчетов | Report() | **+** |

# ВЫВОДЫ

В результате выполнения курсового проекта была разработана корпоративная система для гостиницы с учетом дополнительными услугами, позволяющая вносить в базу данных информацию о посетителей и заказах, а именно заносить информацию в базу данных о том, какой заказ с учетом дополнительных услугами, также каким клиентом (посетителей), на определенный срок и сумму.

Данная корпоративная система предназначена для гостиницы, включает в себя базу данных содержащую информацию об предоставление конкретной услуги конкретному клиенту на ограниченный срок использования.

В процессе подготовки к разработке системы «Гостиницы» проведен анализ требований: мандатные требования; функциональные требования; нефункциональные требования; требования к ПО.

Созданы сценарии использования и ТЗ. Выдвинуты и реализованы требования к надежности.

Реализовано систему учета предоставления услуг на языке программирования С# под платформой Visual Studio 2012 с подключенной СУБД MySQL Server.

Проведено интеграционное и системное тестирования. В результате тестирования исправлены все выявленные ошибки.

Используя знания, полученные при изучении таких дисциплин как «Программное обеспечение корпоративных информационных систнем», «ООП», «Качество и тестирование ПП», «Требования к ПП» была разработана система «Гостиницы». Также были улучшены навыки работы с объектно-ориентированным языком программирования С# и освоены технологию MVC 4.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Автоматизированные системы управления гостиницами. турFAQnet [Электронный ресурс]. / Режим доступа: или URb: http://tourfaq.net/hotel-business/avtomatizirovannye-sistemy-upravleniya-gostinicami/ - 05.02.16.
2. Автоматизированные системы управления гостиницами. [Электронный ресурс]. / Режим доступа: или URb: http://tourfaq.net/hotel-business/avtomatizirovannye-sistemy-upravleniya-gostinicami/ - 15.02.16.
3. Курсовая работа: Разработка автоматизированной информационной системы. Система учета ОАО "ЮТК": [Электронный ресурс]. / Режим доступа: или URb: https://domashke.com/referati/referaty-po-informatike-programmirovaniyu/kursovaya-rabota-razrabotka-avtomatizirovannoj-informacionnoj-sistemy-sistema-ucheta-oao-quotyutkquot - 15.02.16.
4. Обоснование выбора средств разработки [Электронный ресурс]. / Режим доступа: или URb: http://mybiblioteka.su/8-147207.html - 17.02.16.
5. Тамрле, Л. Введение в тестирование программного обеспечения.: Пер. с англ.. – М.: Издательский дом "Вильямс", 2003. – 368 с.

# ПРИЛОЖЕНИЕ А

**ER-диаграмма**

Заказ

Номер

Список услуг

Клиенты

1:1

Тип услуг

Тип комнаты

Тариф номеров

1:m

1:m

1:m

1:m

1:m

Рисунок А.1 – ER-диаграмма

# ПРИЛОЖЕНИЕ Б

**Листинг SQL-запросов построения физической модели данных**

CREATE TABLE [dbo].[Client] (

[Id\_client] INT IDENTITY (1, 1) NOT NULL,

[FIO] NVARCHAR (150) NULL,

[birthday] DATE NULL,

[pol] NCHAR (10) NULL,

[identif] INT NULL,

[passport] NCHAR (100) NULL,

[address] NTEXT NULL,

[phone] INT NULL,

PRIMARY KEY CLUSTERED ([Id\_client] ASC)

);

CREATE TABLE [dbo].[list\_add\_services] (

[Id\_list\_add\_services] INT IDENTITY (1, 1) NOT NULL,

[id\_order] INT NULL,

[id\_servies] INT NULL,

PRIMARY KEY CLUSTERED ([Id\_list\_add\_services] ASC),

CONSTRAINT [FK\_list\_add\_services\_ToTypeServise] FOREIGN KEY ([id\_servies]) REFERENCES [dbo].[type\_servies] ([Id\_servies])

);

CREATE TABLE [dbo].[Orders] (

[Id\_order] INT IDENTITY (1, 1) NOT NULL,

[date] DATE NULL,

[begin] DATETIME NULL,

[end] DATETIME NULL,

[payment] BIT DEFAULT ((0)) NULL,

[id\_client] INT NULL,

[id\_number] INT NULL,

[id\_list\_add\_servies] INT NULL,

PRIMARY KEY CLUSTERED ([Id\_order] ASC),

CONSTRAINT [FK\_Orders\_ToClient] FOREIGN KEY ([id\_client]) REFERENCES [dbo].[Client] ([Id\_client]),

CONSTRAINT [FK\_Orders\_ToNumber] FOREIGN KEY ([id\_number]) REFERENCES [dbo].[type\_number] ([Id\_number]),

CONSTRAINT [FK\_Orders\_ToServies] FOREIGN KEY ([id\_list\_add\_servies]) REFERENCES [dbo].[list\_add\_services] ([Id\_list\_add\_services])

);

CREATE TABLE [dbo].[type\_number] (

[Id\_number] INT IDENTITY (1, 1) NOT NULL,

[name] NCHAR (100) NULL,

[col] INT NULL,

[id\_pool] INT NULL,

PRIMARY KEY CLUSTERED ([Id\_number] ASC),

CONSTRAINT [FK\_type\_number\_ToTypePool] FOREIGN KEY ([id\_pool]) REFERENCES [dbo].[type\_pool] ([Id\_pool])

);

CREATE TABLE [dbo].[type\_pool] (

[Id\_pool] INT IDENTITY (1, 1) NOT NULL,

[type] NCHAR (100) NULL,

[tariff] INT NULL,

PRIMARY KEY CLUSTERED ([Id\_pool] ASC)

);

CREATE TABLE [dbo].[type\_servies] (

[Id\_servies] INT IDENTITY (1, 1) NOT NULL,

[name] NTEXT NULL,

[tariff] INT NULL,

PRIMARY KEY CLUSTERED ([Id\_servies] ASC)

);

# ПРИЛОЖЕНИЕ В

**Листинги исходного кода на языке C#.**

**Листинг В.1 – исходный код AccountController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Transactions;

using System.Web;

using System.Web.Mvc;

using System.Web.Security;

using DotNetOpenAuth.AspNet;

using Microsoft.Web.WebPages.OAuth;

using WebMatrix.WebData;

using MvcApplication1.Filters;

using MvcApplication1.Models;

namespace MvcApplication1.Controllers

{

[Authorize]

[InitializeSimpleMembership]

public class AccountController : Controller

{

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Login(LoginModel model, string returnUrl)

{

if (ModelState.IsValid && WebSecurity.Login(model.UserName, model.Password, persistCookie: model.RememberMe))

{

return RedirectToLocal(returnUrl);

}

// Появление этого сообщения означает наличие ошибки; повторное отображение формы

ModelState.AddModelError("", "Имя пользователя или пароль указаны неверно.");

return View(model);

}

//

// POST: /Account/LogOff

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

WebSecurity.Logout();

return RedirectToAction("Index", "Home");

}

//

// GET: /Account/Register

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult Register(RegisterModel model)

{

if (ModelState.IsValid)

{

// Попытка зарегистрировать пользователя

try

{

WebSecurity.CreateUserAndAccount(model.UserName, model.Password);

WebSecurity.Login(model.UserName, model.Password);

return RedirectToAction("Index", "Home");

}

catch (MembershipCreateUserException e)

{

ModelState.AddModelError("", ErrorCodeToString(e.StatusCode));

}

}

// Появление этого сообщения означает наличие ошибки; повторное отображение формы

return View(model);

}

//

// POST: /Account/Disassociate

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Disassociate(string provider, string providerUserId)

{

string ownerAccount = OAuthWebSecurity.GetUserName(provider, providerUserId);

ManageMessageId? message = null;

// Удалять связь учетной записи, только если текущий пользователь — ее владелец

if (ownerAccount == User.Identity.Name)

{

// Транзакция используется, чтобы помешать пользователю удалить учетные данные последнего входа

using (var scope = new TransactionScope(TransactionScopeOption.Required, new TransactionOptions { IsolationLevel = IsolationLevel.Serializable }))

{

bool hasLocalAccount = OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

if (hasLocalAccount || OAuthWebSecurity.GetAccountsFromUserName(User.Identity.Name).Count > 1)

{

OAuthWebSecurity.DeleteAccount(provider, providerUserId);

scope.Complete();

message = ManageMessageId.RemoveLoginSuccess;

}

}

}

return RedirectToAction("Manage", new { Message = message });

}

//

// GET: /Account/Manage

public ActionResult Manage(ManageMessageId? message)

{

ViewBag.StatusMessage =

message == ManageMessageId.ChangePasswordSuccess ? "Пароль изменен."

: message == ManageMessageId.SetPasswordSuccess ? "Пароль задан."

: message == ManageMessageId.RemoveLoginSuccess ? "Внешняя учетная запись удалена."

: "";

ViewBag.HasLocalPassword = OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

ViewBag.ReturnUrl = Url.Action("Manage");

return View();

}

//

// POST: /Account/Manage

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Manage(LocalPasswordModel model)

{

bool hasLocalAccount = OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

ViewBag.HasLocalPassword = hasLocalAccount;

ViewBag.ReturnUrl = Url.Action("Manage");

if (hasLocalAccount)

{

if (ModelState.IsValid)

{

// В ряде случаев при сбое ChangePassword породит исключение, а не вернет false.

bool changePasswordSucceeded;

try

{

changePasswordSucceeded = WebSecurity.ChangePassword(User.Identity.Name, model.OldPassword, model.NewPassword);

}

catch (Exception)

{

changePasswordSucceeded = false;

}

if (changePasswordSucceeded)

{

return RedirectToAction("Manage", new { Message = ManageMessageId.ChangePasswordSuccess });

}

else

{

ModelState.AddModelError("", "Неправильный текущий пароль или недопустимый новый пароль.");

}

}

}

else

{

// У пользователя нет локального пароля, уберите все ошибки проверки, вызванные отсутствующим

// полем OldPassword

ModelState state = ModelState["OldPassword"];

if (state != null)

{

state.Errors.Clear();

}

if (ModelState.IsValid)

{

try

{

WebSecurity.CreateAccount(User.Identity.Name, model.NewPassword);

return RedirectToAction("Manage", new { Message = ManageMessageId.SetPasswordSuccess });

}

catch (Exception e)

{

ModelState.AddModelError("", e);

}

}

}

// Появление этого сообщения означает наличие ошибки; повторное отображение формы

return View(model);

}

//

// POST: /Account/ExternalLogin

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider, string returnUrl)

{

return new ExternalLoginResult(provider, Url.Action("ExternalLoginCallback", new { ReturnUrl = returnUrl }));

}

//

// GET: /Account/ExternalLoginCallback

[AllowAnonymous]

public ActionResult ExternalLoginCallback(string returnUrl)

{

AuthenticationResult result = OAuthWebSecurity.VerifyAuthentication(Url.Action("ExternalLoginCallback", new { ReturnUrl = returnUrl }));

if (!result.IsSuccessful)

{

return RedirectToAction("ExternalLoginFailure");

}

if (OAuthWebSecurity.Login(result.Provider, result.ProviderUserId, createPersistentCookie: false))

{

return RedirectToLocal(returnUrl);

}

if (User.Identity.IsAuthenticated)

{

// Если текущий пользователь вошел в систему, добавляется новая учетная запись

OAuthWebSecurity.CreateOrUpdateAccount(result.Provider, result.ProviderUserId, User.Identity.Name);

return RedirectToLocal(returnUrl);

}

else

{

// Новый пользователь, запрашиваем желаемое имя участника

string loginData = OAuthWebSecurity.SerializeProviderUserId(result.Provider, result.ProviderUserId);

ViewBag.ProviderDisplayName = OAuthWebSecurity.GetOAuthClientData(result.Provider).DisplayName;

ViewBag.ReturnUrl = returnUrl;

return View("ExternalLoginConfirmation", new RegisterExternalLoginModel { UserName = result.UserName, ExternalLoginData = loginData });

}

}

//

// POST: /Account/ExternalLoginConfirmation

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLoginConfirmation(RegisterExternalLoginModel model, string returnUrl)

{

string provider = null;

string providerUserId = null;

if (User.Identity.IsAuthenticated || !OAuthWebSecurity.TryDeserializeProviderUserId(model.ExternalLoginData, out provider, out providerUserId))

{

return RedirectToAction("Manage");

}

if (ModelState.IsValid)

{

// Добавление нового пользователя в базу данных

using (UsersContext db = new UsersContext())

{

UserProfile user = db.UserProfiles.FirstOrDefault(u => u.UserName.ToLower() == model.UserName.ToLower());

// Проверка наличия пользователя в базе данных

if (user == null)

{

// Добавление имени в таблицу профиля

db.UserProfiles.Add(new UserProfile { UserName = model.UserName });

db.SaveChanges();

OAuthWebSecurity.CreateOrUpdateAccount(provider, providerUserId, model.UserName);

OAuthWebSecurity.Login(provider, providerUserId, createPersistentCookie: false);

return RedirectToLocal(returnUrl);

}

else

{

ModelState.AddModelError("UserName", "Имя пользователя уже существует. Введите другое имя пользователя.");

}

}

}

ViewBag.ProviderDisplayName = OAuthWebSecurity.GetOAuthClientData(provider).DisplayName;

ViewBag.ReturnUrl = returnUrl;

return View(model);

}

//

// GET: /Account/ExternalLoginFailure

[AllowAnonymous]

public ActionResult ExternalLoginFailure()

{

return View();

}

[AllowAnonymous]

[ChildActionOnly]

public ActionResult ExternalLoginsList(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return PartialView("\_ExternalLoginsListPartial", OAuthWebSecurity.RegisteredClientData);

}

[ChildActionOnly]

public ActionResult RemoveExternalLogins()

{

ICollection<OAuthAccount> accounts = OAuthWebSecurity.GetAccountsFromUserName(User.Identity.Name);

List<ExternalLogin> externalLogins = new List<ExternalLogin>();

foreach (OAuthAccount account in accounts)

{

AuthenticationClientData clientData = OAuthWebSecurity.GetOAuthClientData(account.Provider);

externalLogins.Add(new ExternalLogin

{

Provider = account.Provider,

ProviderDisplayName = clientData.DisplayName,

ProviderUserId = account.ProviderUserId,

});

}

ViewBag.ShowRemoveButton = externalLogins.Count > 1 || OAuthWebSecurity.HasLocalAccount(WebSecurity.GetUserId(User.Identity.Name));

return PartialView("\_RemoveExternalLoginsPartial", externalLogins);

}

#region Вспомогательные методы

private ActionResult RedirectToLocal(string returnUrl)

{

if (Url.IsLocalUrl(returnUrl))

{

return Redirect(returnUrl);

}

else

{

return RedirectToAction("Index", "Home");

}

}

public enum ManageMessageId

{

ChangePasswordSuccess,

SetPasswordSuccess,

RemoveLoginSuccess,

}

internal class ExternalLoginResult : ActionResult

{

public ExternalLoginResult(string provider, string returnUrl)

{

Provider = provider;

ReturnUrl = returnUrl;

}

public string Provider { get; private set; }

public string ReturnUrl { get; private set; }

public override void ExecuteResult(ControllerContext context)

{

OAuthWebSecurity.RequestAuthentication(Provider, ReturnUrl);

}

}

private static string ErrorCodeToString(MembershipCreateStatus createStatus)

{

// Полный список кодов состояния см. по адресу http://go.microsoft.com/fwlink/?LinkID=177550

//.

switch (createStatus)

{

case MembershipCreateStatus.DuplicateUserName:

return "Имя пользователя уже существует. Введите другое имя пользователя.";

case MembershipCreateStatus.DuplicateEmail:

return "Имя пользователя для данного адреса электронной почты уже существует. Введите другой адрес электронной почты.";

case MembershipCreateStatus.InvalidPassword:

return "Указан недопустимый пароль. Введите допустимое значение пароля.";

case MembershipCreateStatus.InvalidEmail:

return "Указан недопустимый адрес электронной почты. Проверьте значение и повторите попытку.";

case MembershipCreateStatus.InvalidAnswer:

return "Указан недопустимый ответ на вопрос для восстановления пароля. Проверьте значение и повторите попытку.";

case MembershipCreateStatus.InvalidQuestion:

return "Указан недопустимый вопрос для восстановления пароля. Проверьте значение и повторите попытку.";

case MembershipCreateStatus.InvalidUserName:

return "Указано недопустимое имя пользователя. Проверьте значение и повторите попытку.";

case MembershipCreateStatus.ProviderError:

return "Поставщик проверки подлинности вернул ошибку. Проверьте введенное значение и повторите попытку. Если проблему устранить не удастся, обратитесь к системному администратору.";

case MembershipCreateStatus.UserRejected:

return "Запрос создания пользователя был отменен. Проверьте введенное значение и повторите попытку. Если проблему устранить не удастся, обратитесь к системному администратору.";

default:

return "Произошла неизвестная ошибка. Проверьте введенное значение и повторите попытку. Если проблему устранить не удастся, обратитесь к системному администратору.";

}

}

#endregion

}

}

**Листинг В.2 – HomeController.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using MvcApplication1.Models;

using System.Data;

using System.Data.SqlTypes;

namespace MvcApplication1.Controllers

{

public class HomeController : Controller

{

int number\_ordr\_current;

List<type\_pool> Type\_Room;

public ActionResult Index()

{

return View();

}

public HomeController()

{

Type\_Room = FromDB<type\_pool>("SELECT \* FROM [type\_pool]");

ViewBag.flagTypeRoom = false;

ViewBag.flagColMest = false;

ViewBag.flagNumber = false;

ViewBag.flagCol = false;

ViewBag.flagDateBegin = false;

ViewBag.flagTimeBegin = false;

ViewBag.flagDateEnd = false;

ViewBag.flagTimeEnd = false;

ViewBag.flagPrice = false;

ViewBag.flagSmallButton = false;

ViewBag.flagBigButton = false;

ViewBag.Id\_client = null;

ViewBag.Name\_client = null;

ViewBag.Id\_pool = null;

ViewBag.Id\_number = null;

ViewBag.Col = null;

ViewBag.DateBegin = null;

ViewBag.TimeBegin = null;

ViewBag.DateEnd = null;

ViewBag.TimeEnd = null;

ViewBag.Price = null;

ViewBag.flagServiesDate = false;

ViewBag.flagTimeUse = false;

ViewBag.flagColUse = false;

ViewBag.flagPriceServies = false;

ViewBag.flagServiesButton = false;

ViewBag.IDServies = null;

ViewBag.ServiesDate = null;

ViewBag.TimeUse = null;

ViewBag.ColUse = null;

ViewBag.PriceServies = null;

}

public ActionResult Details(int id)

{

DataClasses1DataContext db = new DataClasses1DataContext();

OrderAndServies orderservies = new OrderAndServies();

var order = db.ExecuteQuery<OrderModific>(@"SELECT [Orders].[id\_client], [Orders].[Id\_order], [Orders].[date], [Orders].[begin], [Orders].[end], [Orders].[payment], [Orders].[BigPrice], [Client].[FIO] FROM [Orders], [Client] WHERE Orders.Id\_order = '" + id + "' AND Orders.id\_client = Client.Id\_client").ToList<OrderModific>().First();

List<list\_add\_serviecsModific> listServies = FromDB<list\_add\_serviecsModific>(@"SELECT [list\_add\_services].\*, [type\_servies].[name] AS name\_servies FROM [list\_add\_services], [type\_servies] WHERE [list\_add\_services].[id\_order] = " + id + " AND [list\_add\_services].id\_servies = [type\_servies].[Id\_servies] ");

List<list\_numberModifc> listNumber = FromDB<list\_numberModifc>(@"SELECT CAST([type\_number].[name] AS NVARCHAR(MAX)) AS 'number\_room', [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)) AS 'type\_room', CAST([nametyperoom].[name] AS NVARCHAR(MAX)) AS 'col\_in\_room', CAST([list\_number].[price] AS NVARCHAR(MAX)) AS 'price' FROM [list\_number], [type\_pool], [type\_number], [nametyperoom] WHERE [list\_number].[id\_order] = " + id + " AND [list\_number].[id\_number\_room] = [type\_number].[id\_number] AND [list\_number].[id\_type\_room] = [type\_pool].Id\_pool AND [list\_number].[id\_col\_in\_room] = [nametyperoom].[Id] GROUP BY CAST([type\_number].[name] AS NVARCHAR(MAX)), [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)), CAST([nametyperoom].[name] AS NVARCHAR(MAX)), CAST([list\_number].[price] AS NVARCHAR(MAX))");

orderservies.order = order;

orderservies.ListServirecs = listServies;

orderservies.ListNumber = listNumber;

return View(orderservies);

}

public ActionResult About()

{

ViewBag.Message = "Страница описания приложения.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Страница контактов.";

return View();

}

public ActionResult CreateOrder(Orders order)

{

var date = DateTime.Now;

{ }

//Создание новый заказ

DataClasses1DataContext db = new DataClasses1DataContext();

Orders item = new Orders()

{

id\_client = Convert.ToInt32(order.id\_client),

date = DateTime.Now,

Done = false,

Hide = false

};

db.Orders.InsertOnSubmit(item);

db.SubmitChanges();

List<Orders> fff = FromDB<Orders>(@"SELECT [Orders].[Id\_order] FROM [Orders]");

string fff222 = fff.Select(it => it.Id\_order).Max().ToString();

ViewBag.IdOrder = fff222;

List<Client> clientFromDB = FromDB<Client>(@"SELECT [Client].[id\_client], [Client].[FIO] FROM [Client]");

ViewBag.flagTypeRoom = true;

ViewBag.Id\_client = order.id\_client;

ViewBag.Name\_client = (from p in clientFromDB where p.Id\_client == order.id\_client select p.FIO).First();

ComplexModelClientAndServies list = new ComplexModelClientAndServies();

List<type\_servies> TypeServiesFromDB = FromDB<type\_servies>(@"SELECT \* FROM [type\_servies]");

List<type\_pool> TypePoolFromDB = FromDB<type\_pool>(@"SELECT \* FROM [type\_pool]");

List<nametyperoom> NameTypeRoomFromDB = FromDB<nametyperoom>(@"SELECT \* FROM [nametyperoom]");

List<type\_number> NumberFromDB = FromDB<type\_number>(@"SELECT \* FROM [type\_number] WHERE [type\_number].[Free] = 1");

//Список подзаказов номеров

List<list\_numberModifc> listNumberOrder = FromDB<list\_numberModifc>(@"SELECT CAST([type\_number].[name] AS NVARCHAR(MAX)) AS 'number\_room', [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)) AS 'type\_room', CAST([nametyperoom].[name] AS NVARCHAR(MAX)) AS 'col\_in\_room', CAST([list\_number].[price] AS NVARCHAR(MAX)) AS 'price' FROM [list\_number], [type\_pool], [type\_number], [nametyperoom] WHERE [list\_number].[id\_order] = " + ViewBag.IdOrder + " AND [list\_number].[id\_number\_room] = [type\_number].[id\_number] AND [list\_number].[id\_type\_room] = [type\_pool].Id\_pool AND [list\_number].[id\_col\_in\_room] = [nametyperoom].[Id] GROUP BY CAST([type\_number].[name] AS NVARCHAR(MAX)), [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)), CAST([nametyperoom].[name] AS NVARCHAR(MAX)), CAST([list\_number].[price] AS NVARCHAR(MAX))");

list.ListClient = clientFromDB;

list.ListTypePool = TypePoolFromDB;

list.ListNameTypeRoom = NameTypeRoomFromDB;

list.ListNumber = NumberFromDB;

list.ListTypeServies = TypeServiesFromDB;

list.llll = listNumberOrder;

list.add\_llll = listAddServiesOrder;

return View("Create", list);

}

public ActionResult Create()

{

ComplexModelClientAndServies list = new ComplexModelClientAndServies();

list.ListClient = FromDB<Client>(@"SELECT [Client].[id\_client], [Client].[FIO] FROM [Client]"); ;

return View(list);

}

public ActionResult ChoiceClient(int Id\_client)

{

int Id\_client111 = Id\_client;

var date = DateTime.Now;

//Создание новый заказ

DataClasses1DataContext db = new DataClasses1DataContext();

Orders item = new Orders()

{

id\_client = Convert.ToInt32(Id\_client111),

date = DateTime.Now,

Done = false,

Hide = false

};

db.Orders.InsertOnSubmit(item);

db.SubmitChanges();

List<Orders> fff = FromDB<Orders>(@"SELECT [Orders].[Id\_order] FROM [Orders]");

string fff222 = fff.Select(it => it.Id\_order).Max().ToString();

ViewBag.IdOrder = fff222;

ViewBag.flagTypeRoom = true;

ViewBag.Id\_client = Id\_client111;

ComplexModelClientAndServies list = new ComplexModelClientAndServies();

List<Client> clientFromDB = FromDB<Client>(@"SELECT [Client].[id\_client], [Client].[FIO] FROM [Client]");

ViewBag.Name\_client = (from p in clientFromDB where p.Id\_client == Id\_client111 select p.FIO).First();

List<type\_servies> TypeServiesFromDB = FromDB<type\_servies>(@"SELECT \* FROM [type\_servies]");

List<type\_pool> TypePoolFromDB = FromDB<type\_pool>(@"SELECT \* FROM [type\_pool]");

List<nametyperoom> NameTypeRoomFromDB = FromDB<nametyperoom>(@"SELECT \* FROM [nametyperoom]");

//Список подзаказов номеров

List<list\_numberModifc> listNumberOrder = FromDB<list\_numberModifc>(@"SELECT CAST([type\_number].[name] AS NVARCHAR(MAX)) AS 'number\_room', [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)) AS 'type\_room', CAST([nametyperoom].[name] AS NVARCHAR(MAX)) AS 'col\_in\_room', CAST([list\_number].[price] AS NVARCHAR(MAX)) AS 'price' FROM [list\_number], [type\_pool], [type\_number], [nametyperoom] WHERE [list\_number].[id\_order] = " + ViewBag.IdOrder + " AND [list\_number].[id\_number\_room] = [type\_number].[id\_number] AND [list\_number].[id\_type\_room] = [type\_pool].Id\_pool AND [list\_number].[id\_col\_in\_room] = [nametyperoom].[Id] GROUP BY CAST([type\_number].[name] AS NVARCHAR(MAX)), [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)), CAST([nametyperoom].[name] AS NVARCHAR(MAX)), CAST([list\_number].[price] AS NVARCHAR(MAX))");

//Список услугов

List<list\_add\_serviecsModific> listAddServiesOrder = FromDB<list\_add\_serviecsModific>(@"SELECT [list\_add\_services].\*, [type\_servies].[name] AS name\_servies FROM [list\_add\_services], [type\_servies] WHERE [list\_add\_services].[id\_order] = '" + ViewBag.IdOrder + "' AND [list\_add\_services].[id\_servies] = [type\_servies].[Id\_servies] ");

list.ListClient = clientFromDB;

list.ListTypePool = TypePoolFromDB;

list.ListNameTypeRoom = NameTypeRoomFromDB;

list.ListTypeServies = TypeServiesFromDB;

list.llll = listNumberOrder;

list.add\_llll = listAddServiesOrder;

return View("Create", list);

}

public ActionResult ProcessingOrder(OrderServiesModific order, int? IdOrder111 )

{

ViewBag.IdOrder = IdOrder111;

ComplexModelClientAndServies list = new ComplexModelClientAndServies();

List<Client> clientFromDB = FromDB<Client>(@"SELECT [Client].[id\_client], [Client].[FIO] FROM [Client]");

ViewBag.Name\_client = (from p in clientFromDB where p.Id\_client == order.Id\_client select p.FIO).First();

List<type\_servies> TypeServiesFromDB = FromDB<type\_servies>(@"SELECT \* FROM [type\_servies]");

List<type\_pool> TypePoolFromDB = FromDB<type\_pool>(@"SELECT \* FROM [type\_pool]");

List<nametyperoom> NameTypeRoomFromDB = FromDB<nametyperoom>(@"SELECT \* FROM [nametyperoom]");

ViewBag.IdOrder = IdOrder111;

List<list\_numberModifc> listNumberOrder = FromDB<list\_numberModifc>(@"SELECT CAST([type\_number].[name] AS NVARCHAR(MAX)) AS 'number\_room', [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)) AS 'type\_room', CAST([nametyperoom].[name] AS NVARCHAR(MAX)) AS 'col\_in\_room', CAST([list\_number].[price] AS NVARCHAR(MAX)) AS 'price' FROM [list\_number], [type\_pool], [type\_number], [nametyperoom] WHERE [list\_number].[id\_order] = " + IdOrder111 + " AND [list\_number].[id\_number\_room] = [type\_number].[id\_number] AND [list\_number].[id\_type\_room] = [type\_pool].Id\_pool AND [list\_number].[id\_col\_in\_room] = [nametyperoom].[Id] GROUP BY CAST([type\_number].[name] AS NVARCHAR(MAX)), [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)), CAST([nametyperoom].[name] AS NVARCHAR(MAX)), CAST([list\_number].[price] AS NVARCHAR(MAX))");

//Список услугов

List<list\_add\_serviecsModific> listAddServiesOrder = FromDB<list\_add\_serviecsModific>(@"SELECT [list\_add\_services].\*, [type\_servies].[name] AS name\_servies FROM [list\_add\_services], [type\_servies] WHERE [list\_add\_services].[id\_order] = " + IdOrder111 + " AND [list\_add\_services].id\_servies = [type\_servies].[Id\_servies] ");

list.ListClient = clientFromDB;

list.ListTypePool = TypePoolFromDB;

list.ListNameTypeRoom = NameTypeRoomFromDB;

list.ListTypeServies = TypeServiesFromDB;

list.llll = listNumberOrder;

list.add\_llll = listAddServiesOrder;

if (order.Id\_client != null)

{

ViewBag.flagTypeRoom = true;

ViewBag.Id\_client = order.Id\_client;

ViewBag.Name\_client = (from p in clientFromDB where p.Id\_client == order.Id\_client select p.FIO).First();

}

if (order.Id\_pool != null)

{

ViewBag.flagColMest = true;

ViewBag.Id\_pool = order.Id\_pool;

}

if (order.Id != null)

{

ViewBag.flagNumber = true;

ViewBag.Id = order.Id;

List<type\_number> NumberFromDB = FromDB<type\_number>(@"SELECT \* FROM [type\_number] WHERE [type\_number].[id\_pool] = " + order.Id\_pool + " AND [type\_number].[col]=" + order.Id + " AND [type\_number].[Free] = '1'");

list.ListNumber = NumberFromDB;

//return View("Create", list);

}

if (order.Id\_number != null)

{

ViewBag.flagDateBegin = true;

ViewBag.Begin = order.begin\_date;

}

if (order.begin\_date != null)

{

ViewBag.flagTimeBegin = true;

ViewBag.TimeBegin = order.begin\_date.Value.TimeOfDay;

}

if (order.begin\_time != null)

{

ViewBag.flagDateEnd = true;

ViewBag.DateEnd = order.end\_date;

order.begin\_date = new DateTime(order.begin\_date.Value.Year, order.begin\_date.Value.Month, order.begin\_date.Value.Day, order.begin\_time.Value.Hour, order.begin\_time.Value.Minute, order.begin\_time.Value.Second);

}

if (order.end\_date != null)

{

ViewBag.flagTimeEnd = true;

ViewBag.TimeEnd = order.end\_time;

}

int price;

int tariff;

if (order.end\_time != null)

{

order.end\_date = new DateTime(order.end\_date.Value.Year, order.end\_date.Value.Month, order.end\_date.Value.Day, order.end\_time.Value.Hour, order.end\_time.Value.Minute, order.end\_time.Value.Second);

//Тип комнаты

var ddd333 = Type\_Room.Where(el => el.Id\_pool == order.Id\_pool).FirstOrDefault();

tariff = Convert.ToInt32(ddd333.tariff);

var fff = Type\_Room;

//Дней

DateTime data1 = Convert.ToDateTime(order.begin\_date);

DateTime data2 = Convert.ToDateTime(order.end\_date);

if (data2 > data1)

{

order.begin\_date = data2;

order.end\_date = data1;

}

var date5 = data2 - data1;

int day = date5.Days;

if (day == 0) day = 1;

ViewBag.flagPrice = true;

ViewBag.Price = tariff \* day;

order.price = tariff \* day;

ViewBag.flagSmallButton = true;

}

// приянть подзаказы

if (Request.Form["submitbutton21"] != null)

{

DataClasses1DataContext db = new DataClasses1DataContext();

var item = new list\_number()

{

id\_order = Convert.ToInt32(IdOrder111),

id\_type\_room = Convert.ToInt32(order.Id\_pool),

id\_col\_in\_room = Convert.ToInt32(order.Id),

id\_number\_room = Convert.ToInt32(order.Id\_number),

settlement\_date = Convert.ToDateTime(order.begin\_date),

eviction\_date = Convert.ToDateTime(order.end\_date),

price = Convert.ToInt32(order.price)

};

db.list\_number.InsertOnSubmit(item);

db.SubmitChanges();

type\_number item22 = db.type\_number.Single(e => e.Id\_number == order.Id\_number);

var item2 = new type\_number()

{

id\_pool = order.Id\_number,

Free = !item22.Free

};

ViewBag.flagColMest = false;

ViewBag.flagNumber = false;

ViewBag.flagCol = false;

ViewBag.flagDateBegin = false;

ViewBag.flagTimeBegin = false;

ViewBag.flagDateEnd = false;

ViewBag.flagTimeEnd = false;

ViewBag.flagPrice = false;

ViewBag.flagSmallButton = false;

ViewBag.Id\_pool = null;

ViewBag.Id\_number = null;

ViewBag.Col = null;

ViewBag.DateBegin = null;

ViewBag.TimeBegin = null;

ViewBag.DateEnd = null;

ViewBag.TimeEnd = null;

ViewBag.Price = null;

listNumberOrder = FromDB<list\_numberModifc>(@"SELECT CAST([type\_number].[name] AS NVARCHAR(MAX)) AS 'number\_room', [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)) AS 'type\_room', CAST([nametyperoom].[name] AS NVARCHAR(MAX)) AS 'col\_in\_room', CAST([list\_number].[price] AS NVARCHAR(MAX)) AS 'price' FROM [list\_number], [type\_pool], [type\_number], [nametyperoom] WHERE [list\_number].[id\_order] = " + IdOrder111 + " AND [list\_number].[id\_number\_room] = [type\_number].[id\_number] AND [list\_number].[id\_type\_room] = [type\_pool].Id\_pool AND [list\_number].[id\_col\_in\_room] = [nametyperoom].[Id] GROUP BY CAST([type\_number].[name] AS NVARCHAR(MAX)), [list\_number].[settlement\_date], [list\_number].[eviction\_date], CAST([type\_pool].[type] AS NVARCHAR(MAX)), CAST([nametyperoom].[name] AS NVARCHAR(MAX)), CAST([list\_number].[price] AS NVARCHAR(MAX))");

list.llll = listNumberOrder;

//Список услугов

listAddServiesOrder = FromDB<list\_add\_serviecsModific>(@"SELECT [list\_add\_services].\*, [type\_servies].[name] AS name\_servies FROM [list\_add\_services], [type\_servies] WHERE [list\_add\_services].[id\_order] = " + IdOrder111 + " AND [list\_add\_services].id\_servies = [type\_servies].[Id\_servies] ");

list.add\_llll = listAddServiesOrder;

order.Id\_pool = null;

order.Id = null;

order.Id\_number = null;

ViewBag.Titles = " --- Виберите --- ";

int priceAllOrder = FromDBOne(@"SELECT SUM([list\_number].[price]) FROM [list\_number] WHERE [list\_number].[id\_order] =" + IdOrder111);

int priceAllServiecs = FromDBOne(@"SELECT SUM([list\_add\_services].[price]) FROM [list\_add\_services] WHERE [list\_add\_services].[id\_order] = " + IdOrder111);

list.BigPrice = priceAllOrder + priceAllServiecs;

ViewBag.BigPrice = priceAllOrder + priceAllServiecs;

return View("Create", list);

}

//принять дпо. услуги

if (order.id\_servies != null)

{

ViewBag.flagServiesDate = true;

ViewBag.IDServies = order.id\_servies;

}

if (order.dateUse != null)

{

ViewBag.flagTimeUse = true;

ViewBag.dateUse = order.dateUse;

}

if (order.timeUse != null)

{

ViewBag.flagColUse = true;

ViewBag.timeUse = order.timeUse;

order.dateUse = new DateTime(order.dateUse.Value.Year, order.dateUse.Value.Month, order.dateUse.Value.Day, order.timeUse.Value.Hour, order.timeUse.Value.Minute, order.timeUse.Value.Second);

}

if (order.colServ != null)

{

ViewBag.flagPriceServies = true;

ViewBag.flagServiesButton = true;

ViewBag.ColUse = order.colServ;

var ttt = list.ListTypeServies;

var temp = ttt.Where(el => el.Id\_servies == order.id\_servies).FirstOrDefault();

int tarrif = Convert.ToInt32(temp.tariff);

order.priceServies = order.colServ \* tarrif;

ViewBag.PriceServies = order.priceServies;

}

// полній заказ с услугами

if (Request.Form["submitbuttonServies"] != null)

{

var fff = order;

DataClasses1DataContext db = new DataClasses1DataContext();

list\_add\_services item = new list\_add\_services()

{

id\_order = Convert.ToInt32(IdOrder111),

id\_servies = Convert.ToInt32(order.id\_servies),

DateUse = Convert.ToDateTime(order.dateUse),

col = order.colServ,

price = order.priceServies

};

db.list\_add\_services.InsertOnSubmit(item);

db.SubmitChanges();

ViewBag.flagServiesDate = false;

ViewBag.flagTimeUse = false;

ViewBag.flagColUse = false;

ViewBag.flagPriceServies = false;

ViewBag.flagServiesButton = false;

ViewBag.IDServies = null;

ViewBag.ServiesDate = null;

ViewBag.TimeUse = null;

ViewBag.ColUse = null;

ViewBag.PriceServies = null;

list.add\_llll = FromDB<list\_add\_serviecsModific>(@"SELECT [list\_add\_services].\*, [type\_servies].[name] AS name\_servies FROM [list\_add\_services], [type\_servies] WHERE [list\_add\_services].[id\_order] = " + ViewBag.IdOrder + " AND [list\_add\_services].id\_servies = [type\_servies].[Id\_servies] "); ;

order.id\_servies = null;

int priceAllOrder = FromDBOne(@"SELECT SUM([list\_number].[price]) FROM [list\_number] WHERE [list\_number].[id\_order] =" + IdOrder111);

int priceAllServiecs = FromDBOne(@"SELECT SUM([list\_add\_services].[price]) FROM [list\_add\_services] WHERE [list\_add\_services].[id\_order] = " + IdOrder111);

list.BigPrice = priceAllOrder + priceAllServiecs;

ViewBag.BigPrice = priceAllOrder + priceAllServiecs;

}

if (Request.Form["submitbuttonBigOrder"] != null)

{

int BigPrice = list.BigPrice;

DataClasses1DataContext db = new DataClasses1DataContext();

List<DateTime> trempDateTime = new List<DateTime>();

if (list.llll.Count != 0)

{

foreach (var item in list.llll)

{

trempDateTime.Add(item.settlement\_date);

trempDateTime.Add(item.eviction\_date);

}

DateTime minDate = DateTime.MaxValue;

DateTime maxDate = DateTime.MinValue;

foreach (DateTime dateString in trempDateTime)

{

DateTime date = dateString;

if (date < minDate)

minDate = date;

if (date > maxDate)

maxDate = date;

}

int priceAllOrder = FromDBOne(@"SELECT SUM([list\_number].[price]) FROM [list\_number] WHERE [list\_number].[id\_order] =" + IdOrder111);

int priceAllServiecs = FromDBOne(@"SELECT SUM([list\_add\_services].[price]) FROM [list\_add\_services] WHERE [list\_add\_services].[id\_order] = " + IdOrder111);

int bigPriceeee = priceAllOrder + priceAllServiecs;

Orders item22 = db.Orders.Single(e => e.Id\_order == IdOrder111);

item22.payment = order.payment;

item22.begin = minDate;

item22.end = maxDate;

item22.BigPrice = bigPriceeee;

}

else

{

Orders item22 = db.Orders.Single(e => e.Id\_order == IdOrder111);

item22.begin = null;

item22.end = null;

}

db.SubmitChanges();

db = new DataClasses1DataContext();

List<Order> orders = new List<Order>();

List<list\_add\_services> servies = new List<list\_add\_services>();

orders = db.ExecuteQuery<Order>(@"SELECT \* FROM Orders").ToList<Order>();

var ttt = db.ExecuteQuery<OrderModific>(@"SELECT [Orders].[Id\_order], [Orders].[date], [Orders].[begin], [Orders].[end], [Orders].[payment], [Orders].[BigPrice],[Client].[FIO] FROM [Orders], [Client] WHERE Orders.id\_client = Client.Id\_client;").ToList<OrderModific>();

return Redirect("/Home/Index");

}

return View("Create", list);

}

public ActionResult ShowOrder()

{

DataClasses1DataContext db = new DataClasses1DataContext();

//Очистка Null

SerachClearNull(db, "SELECT \* FROM [Orders] WHERE [Orders].[begin] IS NULL");

//Поулчить список

List<list\_add\_services> servies = new List<list\_add\_services>();

List<Order> orders = new List<Order>();

var ttt = db.ExecuteQuery<OrderModific>(@"SELECT [Orders].[Id\_order], [Orders].[date], [Orders].[begin], [Orders].[end], [Orders].[payment], [Orders].[BigPrice],[Client].[FIO] FROM [Orders], [Client] WHERE Orders.id\_client = Client.Id\_client AND [Orders].[Done] = 'false'").ToList<OrderModific>();

return View(ttt);

}

private static void SerachClearNull(DataClasses1DataContext db, string query)

{

var orders111Begin = db.ExecuteQuery<OrderModific>(query).ToList<OrderModific>();

foreach (var elem in orders111Begin)

{

var item = db.Orders.Single(e => e.Id\_order == elem.id\_order);

db.Orders.DeleteOnSubmit(item);

db.SubmitChanges();

}

}

public ActionResult AddClient(ClientModif client)

{

if (!ModelState.IsValid)

return View("CreateClient");

else

{

DataClasses1DataContext db = new DataClasses1DataContext();

Client item = new Client

{

FIO = client.FIO,

birthday = client.birthday,

address = client.address,

identif = client.identif,

passport = client.passport,

phone = client.phone,

pol = client.pol

};

db.Client.InsertOnSubmit(item);

db.SubmitChanges();

return View("Index");

}

}

public ActionResult Hide(int id)

{

DataClasses1DataContext db = new DataClasses1DataContext();

var item = db.Orders.Single(e => e.Id\_order == id);

item.Done = true;

db.SubmitChanges();

return View("Index");

}

public ActionResult BuyPlay(int tttt)

{

DataClasses1DataContext db = new DataClasses1DataContext();

var item = db.Orders.Single(e => e.Id\_order == tttt);

item.payment = true;

db.SubmitChanges();

return Redirect("/Home/Details/" + tttt);

}

public ActionResult BuyPlayCancel(int tttt)

{

DataClasses1DataContext db = new DataClasses1DataContext();

var item = db.Orders.Single(e => e.Id\_order == tttt);

item.payment = false;

db.SubmitChanges();

return Redirect("/Home/Details/" + tttt);

}

private List<T> FromDB<T>(string query)

{

DataClasses1DataContext db = new DataClasses1DataContext();

var typeservies = db.ExecuteQuery<T>(query).ToList<T>();

return typeservies;

}

private int FromDBOne(string query)

{

DataClasses1DataContext db = new DataClasses1DataContext();

var typeservies = db.ExecuteQuery<int?>(query).First<int?>();

if (typeservies == null) typeservies = 0;

return Convert.ToInt32(typeservies);

}

private string FromDBID<T>(string query)

{

DataClasses1DataContext db = new DataClasses1DataContext();

string id = db.ExecuteQuery<T>(query).First().ToString();

return id;

}

private ComplexModelClientAndServies UpDateData()

{

ComplexModelClientAndServies BigList = new ComplexModelClientAndServies();

return BigList;

}

}

}

**Листинг В.3 – Index.cshtml**

@model IEnumerable<MvcApplication1.Models.OrderModific>

@{

ViewBag.Title = "Домашняя страница";

}

@section featured {

<section class="featured">

<div class="content-wrapper">

<hgroup class="title">

<h1>@ViewBag.Title.</h1>

</hgroup>

<p>

Корпоративная система "Гостиница", версия 1.0.0.0

</p>

</div>

</section>

}

<style>

table

{

border: 1px solid black;

}

td, th

{

border: 1px solid black;

padding:15px;

text-align:center;

}

</style>

@if (Request.IsAuthenticated) {

<text>

Здравствуйте, @Html.ActionLink(User.Identity.Name, "Manage", "Account", routeValues: null, htmlAttributes: new { @class = "username", title = "Управление" })!

@using (Html.BeginForm("LogOff", "Account", FormMethod.Post, new { id = "logoutForm" })) {

@Html.AntiForgeryToken()

<a href="javascript:document.getElementById('logoutForm').submit()">Выйти</a>

}

</text>

<p>

@Html.ActionLink("Новый клиент", "CreateClient", "Home") <br/>

</p>

<p>

@Html.ActionLink("Новый заказ", "Create") <br/>

</p>

<p>

@Html.ActionLink("Регистрация тарифов номеров и дополнительных услуг", "CreateTariff", "Tariff")

</p>

<p>

@Html.ActionLink("Список свободных комнат", "ListRoom", "Room");

</p>

<hr/>

<h2>Текущие заказы:</h2>

<div>

<h3>Список заказов</h3>

</div>

@Html.Action("ShowOrder", "Home");

<div>

@Html.ActionLink("Статистика", "Report", "Report")

@Html.ActionLink("Отчеты", "ReportOut", "Report")

</div>

}

else

{

<h3>Рекомендуются следующие действия:</h3>

<ol class="round">

<li class="one">

<h5>Приступая к работе</h5>

Чтоб приступить работу, то Вам необходимо войти в систему. Если у Вас нет аккаунта, то Вам нужно регистрировать.

</li>

<li class="two">

<h5>Проблемы?</h5>

Если возникли проблемы или неполадки, то свяжитесь с администратором

</li>

<li class="three">

<h5>Тут может быть реклама</h5>

Этот вопрос к менеджеру гостиницы

</li>

</ol>

}

**Листинг В.4 – ShowOrder.cshtml**

@model IEnumerable<MvcApplication1.Models.OrderModific>

@{

Layout = null;

}

<!DOCTYPE html>

<table border="1" style="border:1px solid black; padding:1px;" cellspacing="1" cellpadding="10">

<tr>

<th>

Код заказа

</th>

<th>

ФИО клиента

</th>

<th>

Дата принятия заказа

</th>

<th>

Начало заказа (вселение)

</th>

<th>

Конец заказа (выселение)

</th>

<th>

Общая цена

</th>

<th>

Оплата

</th>

<th>

Завершить заказ

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td align="center">

@Html.DisplayFor(modelItem => item.id\_order)

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

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

</td>

<td>

@Html.ActionLink("Завершить", "Hide", new { id=item.id\_order })

</td>

<td>

@\* @Html.ActionLink("Редактирование", "Edit", new { id=item.id\_order }) | \*@

@Html.ActionLink("Подробности", "Details", new { id=item.id\_order })

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

</td>

</tr>

}

</table>

**Листинг В.5 – Report.cshtml**

@model MvcApplication1.Models.ForReport.ComplexModels

@{

ViewBag.Title = "Report";

}

<h2>Report</h2>

<div>

<table>

<tr>

<td>

<div>

<p><b>Цена дополнительных услуг (грн.):</b></p>

<ul>

@foreach(var item in @Model.listServiecs)

{

<li>@item.name : @item.tariff грн</li>

}

</ul>

<div>

<div>

<p><b>Цена номеров (грн. в сутки):</b></p>

<ul>

@foreach(var item in @Model.listNumber)

{

<li>@item.type : @item.tariff грн</li>

}

</ul>

</div>

</td>

<td>

<div>

<p><b>Итоги:</b></p>

Количество общих заказов: @Model.GeneralOrdersCount<br />

Сумма общих заказов: @Model.GeneralOrdersSum грн.<br />

Количество текущих заказов: @Model.CurrentOrdersCount<br />

Сумма текущих заказов: @Model.CurrentOrdersSum грн.<br />

Количество завершающих заказов: @Model.EndOrdersCount<br />

Сумма завершающих заказов: @Model.EndOrdersSum грн.<br />

Количество неоплачиваемых заказов: @Model.OrdersCountNotPayment<br />

Сумма неоплачивающих заказов: @Model.OrdersSumNotPayment грн.<br />

Количество оплачиваемых заказов: @Model.OrdersCountPayment<br />

Сумма оплачивающих заказов: @Model.OrdersSumPayment грн.<br />

<br/>

<br/>

</div>

</td>

<td>

<div>

<p><b>Кол. предоставлены дополнительных услуг:</b></p>

Предоставили услуг в заказе: @Model.ServiecsOnInOrderCount <br/>

Общий количество услуг: @Model.GeneralServiecs <br/>

Сумма всего услуг: @Model.ServiecsSum грн.<br/>

<br/>

<p><b>Клиенты:</b></p>

Количество регистрационных клиентов: @Model.ClientCount<br/>

"Постоянные" клиенты (5 и более заказов): @Model.ClientConstCount<br/>

Клиенты-халявчики: @Model.ClientNotPaymentCount<br/>

</div>

</td>

</tr>

<tr>

<td>

<b>Список "постоянных" клиентов:</b>

<ol>

@foreach (var item in @Model.listClientConst)

{

<li>@item.FIO</li>

}

</ol>

</td>

<td>

<b>Список уклоняющих клиентов:</b>

<ol>

@foreach (var item in @Model.listClientNotPayment)

{

<li>@item.FIO</li>

}

</ol>

</td>

</tr>

</table>

</div>

**Листинг В.6 – Report.cshtml**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using MvcApplication1.Models.ForReport;

namespace MvcApplication1.Controllers

{

public class ReportController : Controller

{

//

// GET: /Report/

public ActionResult Report()

{

DataClasses1DataContext db = new DataClasses1DataContext();

ComplexModels list = new ComplexModels();

var listTariffServiecs = db.ExecuteQuery<type\_servies>(@"SELECT \* FROM [type\_servies]").ToList<type\_servies>();

var listTariffNumber = db.ExecuteQuery<type\_pool>(@"SELECT \* FROM [type\_pool]").ToList<type\_pool>();

var listOrders = db.ExecuteQuery<Orders>(@"SELECT \* FROM [Orders]").ToList<Orders>();

//Тарификация услуг

list.listServiecs = listTariffServiecs;

//Тарификация номеров

list.listNumber = listTariffNumber;

//Колиечство заказов

list.GeneralOrdersCount = listOrders.Count;

//Сумма общих заказов

list.GeneralOrdersSum = db.ExecuteQuery<int>(@"SELECT SUM([Orders].[BigPrice]) FROM [Orders]").Single();

//Текущие заказы

list.CurrentOrdersCount = db.ExecuteQuery<int>(@"SELECT COUNT([Orders].[Id\_order]) AS 'Id\_order' FROM [Orders] WHERE [Orders].[Done] = 'false'").Single();

//Сумма текущих заказов

list.CurrentOrdersSum = db.ExecuteQuery<int>(@"SELECT SUM([Orders].[Id\_order]) AS 'Id\_order' FROM [Orders] WHERE [Orders].[Done] = 'false'").Single();

//Завершающие заказы

list.EndOrdersCount = db.ExecuteQuery<int>(@"SELECT COUNT([Orders].[Id\_order]) AS 'Id\_order' FROM [Orders] WHERE [Orders].[Done] = 'true'").Single();

//Сумма завершающих заказов

list.EndOrdersSum = db.ExecuteQuery<int>(@"SELECT SUM([Orders].[Id\_order]) AS 'Id\_order' FROM [Orders] WHERE [Orders].[Done] = 'true'").Single();

//Количество неоплачивающих

list.OrdersCountNotPayment = db.ExecuteQuery<int>(@"SELECT COUNT([Orders].[Id\_order]) AS 'Id\_order' FROM [Orders] WHERE [Orders].[payment] = 'false'").Single();

//Неоплачивающие суммы

list.OrdersSumNotPayment = db.ExecuteQuery<int>(@"SELECT SUM([Orders].[Id\_order]) AS 'Id\_order' FROM [Orders] WHERE [Orders].[payment] = 'false'").Single();

//Количество оплачивающих

list.OrdersCountPayment = db.ExecuteQuery<int>(@"SELECT COUNT([Orders].[Id\_order]) AS 'Id\_order' FROM [Orders] WHERE [Orders].[payment] = 'true'").Single();

//Оплачивающие суммы

list.OrdersSumPayment = db.ExecuteQuery<int>(@"SELECT SUM([Orders].[Id\_order]) AS 'Id\_order' FROM [Orders] WHERE [Orders].[payment] = 'true'").Single();

//Предоставили услуг в заказе

var lll = db.ExecuteQuery<StatusServiecs>(@"SELECT [Orders].[Id\_order], COUNT(\*) AS 'ttt' FROM [Orders], [list\_add\_services] WHERE [Orders].[Id\_order] = [list\_add\_services].[id\_order] GROUP BY [Orders].[Id\_order]").ToList<StatusServiecs>();

list.ServiecsOnInOrderCount = lll.Count();

//Общий количество услуг

int inc = 0;

foreach (var item in lll)

{

inc += item.ttt;

}

list.GeneralServiecs = inc;

//Сумма всего услуг

var temp = db.ExecuteQuery<StatusServiecs>("SELECT SUM([list\_add\_services].[price]) AS 'ttt' FROM [list\_add\_services], [type\_servies] WHERE [list\_add\_services].[id\_servies] = [type\_servies].[Id\_servies] GROUP BY [list\_add\_services].[Id\_order]").ToList<StatusServiecs>();

int sum = 0;

foreach (var item in temp)

{

sum += item.ttt;

}

list.ServiecsSum = sum;

// Количество регистрационных клиентов

var listClient = db.ExecuteQuery<Client>("SELECT \* FROM [Client]").ToList<Client>();

list.ClientCount = listClient.Count();

// "Постоянные" клиенты (5 и более заказов)

var temp2 = db.ExecuteQuery<Client>("SELECT [Client].[Id\_client], [Client].[FIO], COUNT([Orders].[Id\_order]) FROM [Orders], [Client] WHERE [Orders].[id\_client] = [Client].[Id\_client] AND [Orders].[payment] = 'true' GROUP BY [Client].[Id\_client], [Client].[FIO] HAVING (COUNT([Orders].[Id\_order])) >= 2").ToList<Client>();

list.ClientConstCount = temp2.Count();

list.listClientConst = temp2;

// Клиенты-халявчики

var temp3 = db.ExecuteQuery<Client>("SELECT [Client].[Id\_client], [Client].[FIO] FROM [Orders], [Client] WHERE [Orders].[id\_client] = [Client].[Id\_client] AND [Orders].[payment] = 'false' GROUP BY [Client].[Id\_client], [Client].[FIO]").ToList<Client>();

list.ClientNotPaymentCount = temp3.Count();

list.listClientNotPayment = temp3;

return View(list);

}

public ActionResult ReportOut()

{

return View();

}

}

class StatusServiecs

{

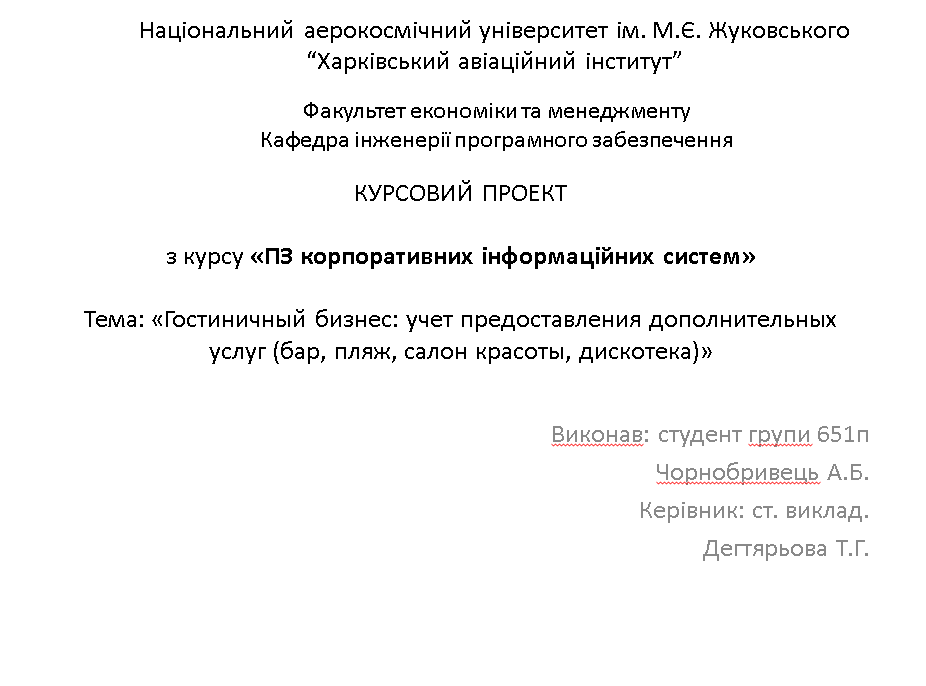
public int ttt;

}

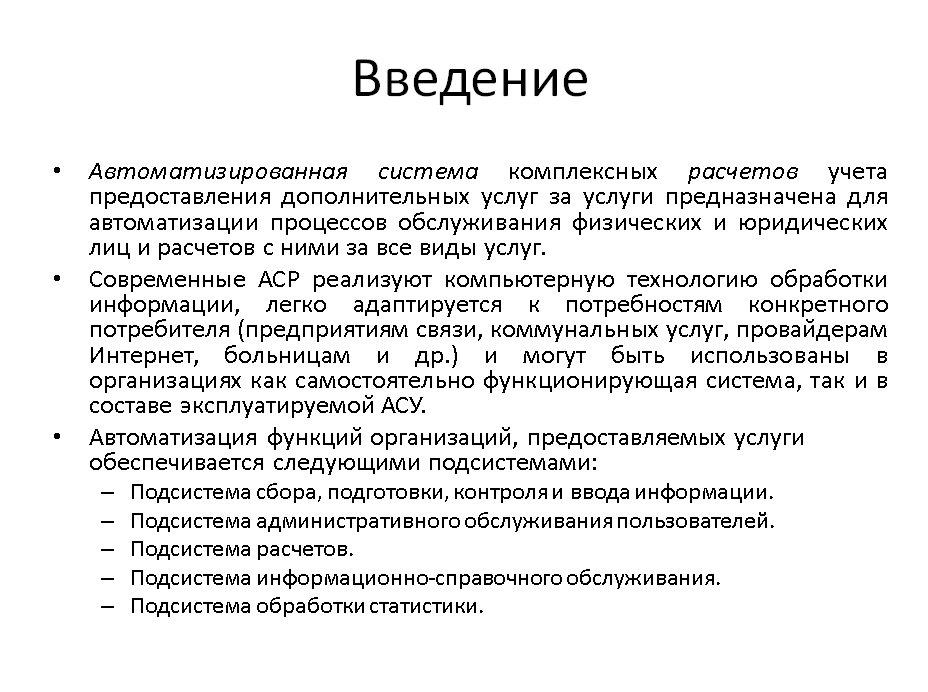
}

# ПРИЛОЖЕНИЕ Г

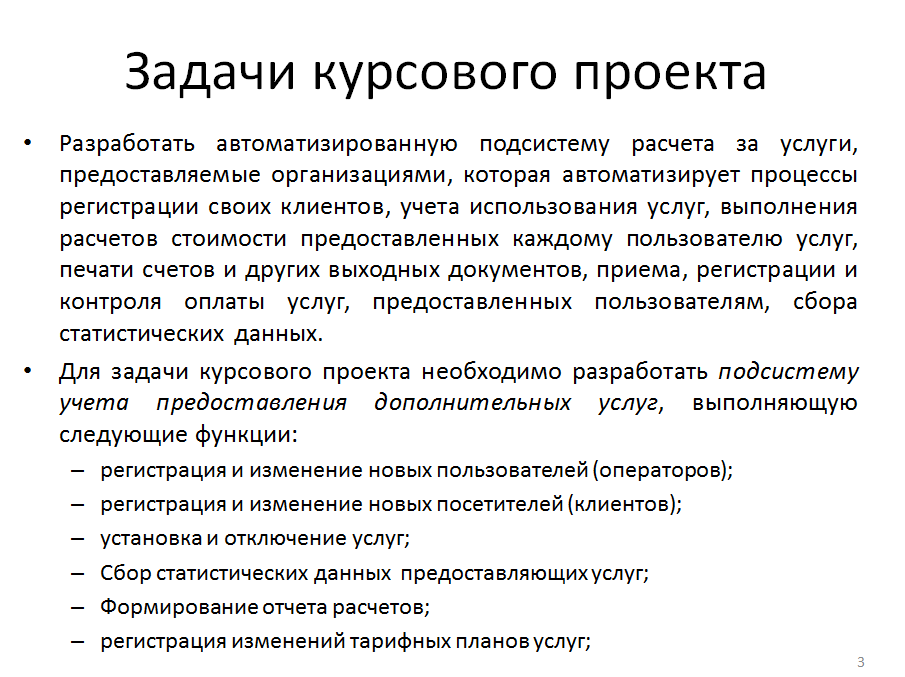
**Слайды презентации**



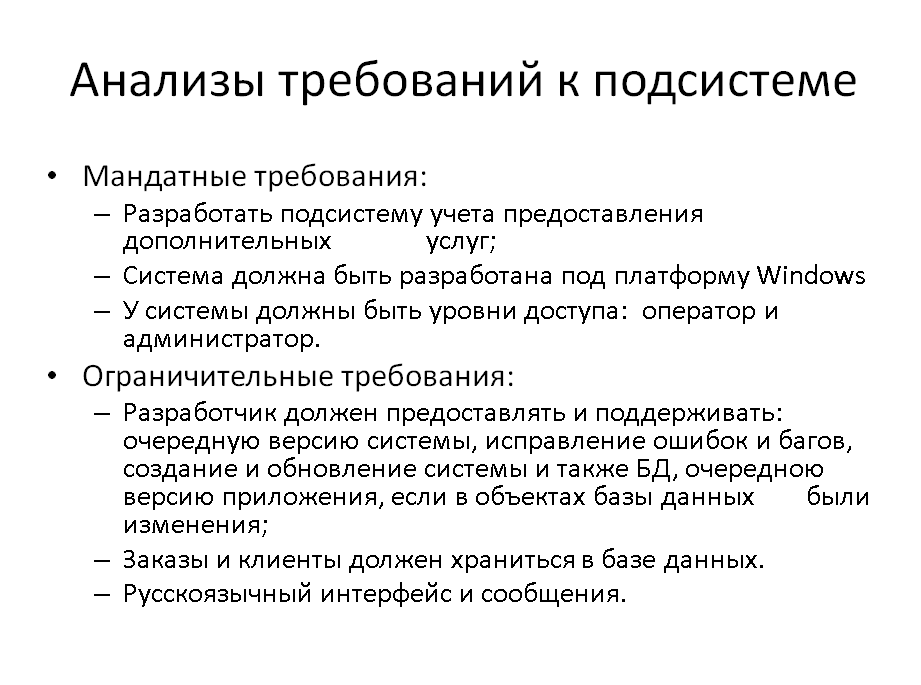
Слайд Г.1 – Титульный лист



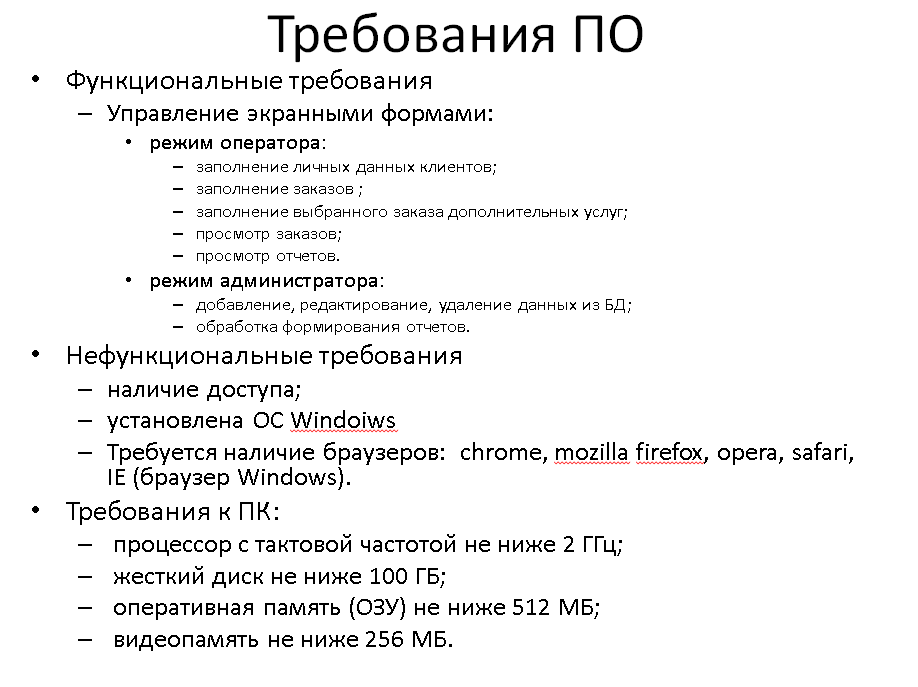
Слайд Г.2 – Введение



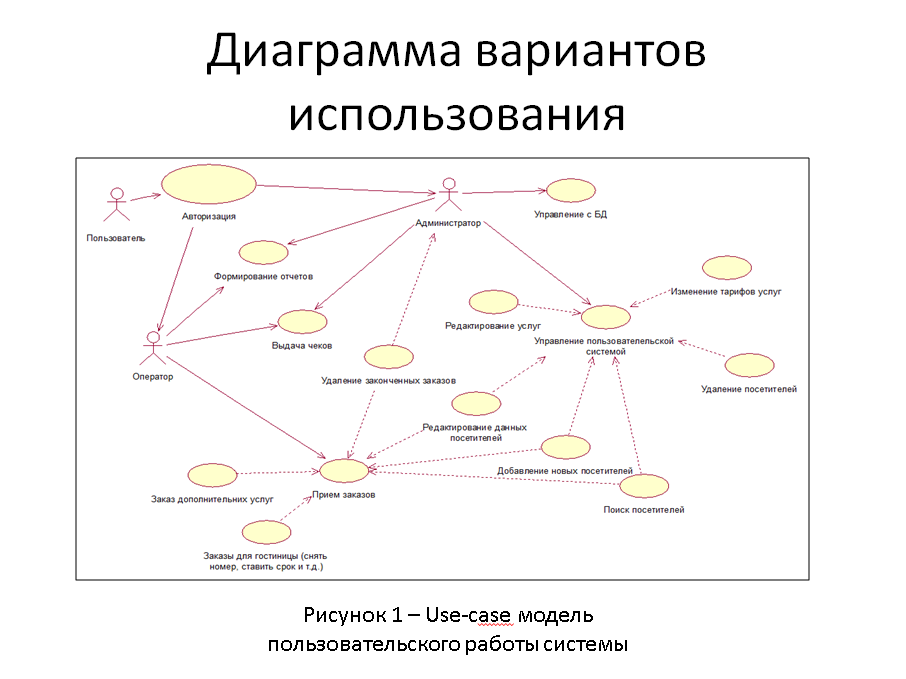
Слайд Г.3 – Задачи проекта



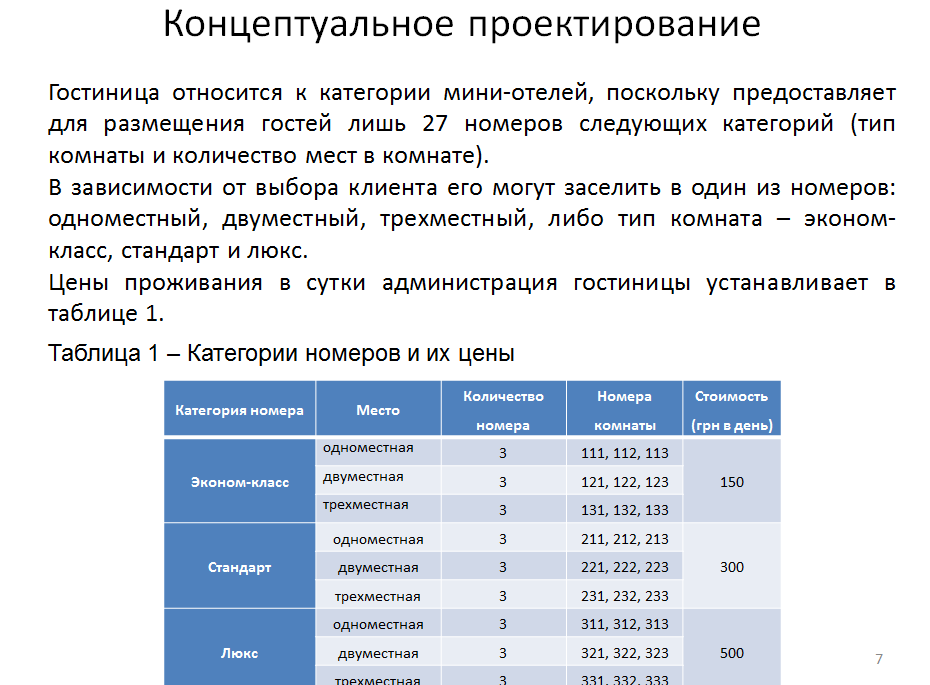
Слайд Г.4 - Анализы требований к подсистеме



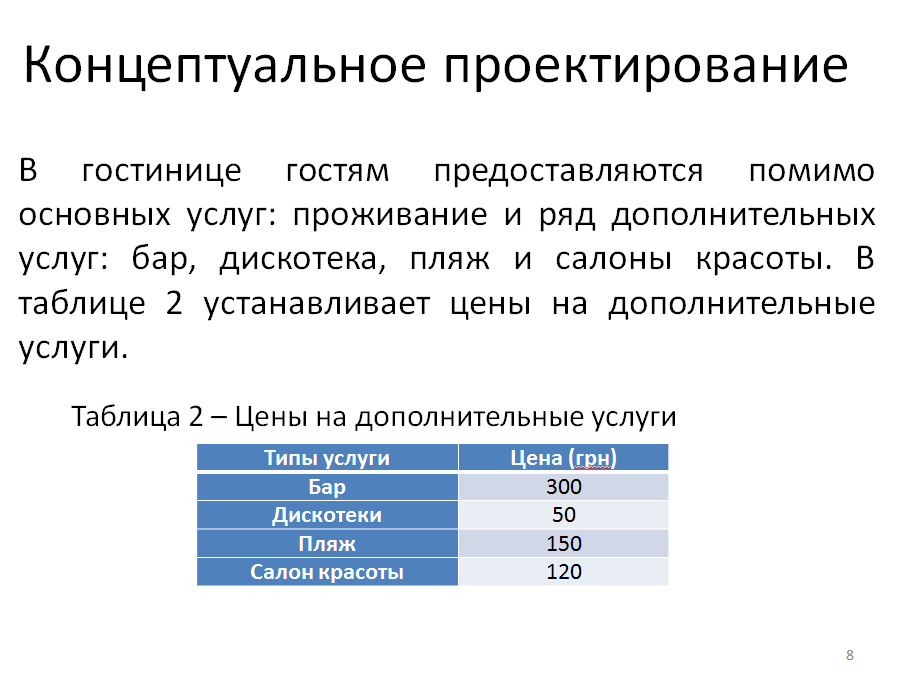
Слайд Г.5 – Требования ПО



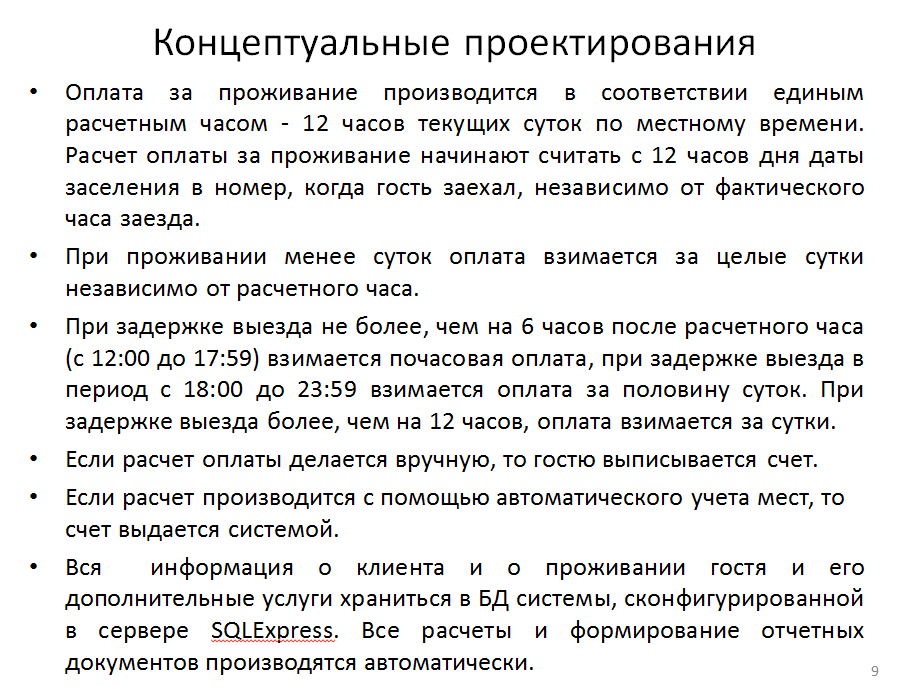
Слайд Г.6 – Диаграмма вариантов использования



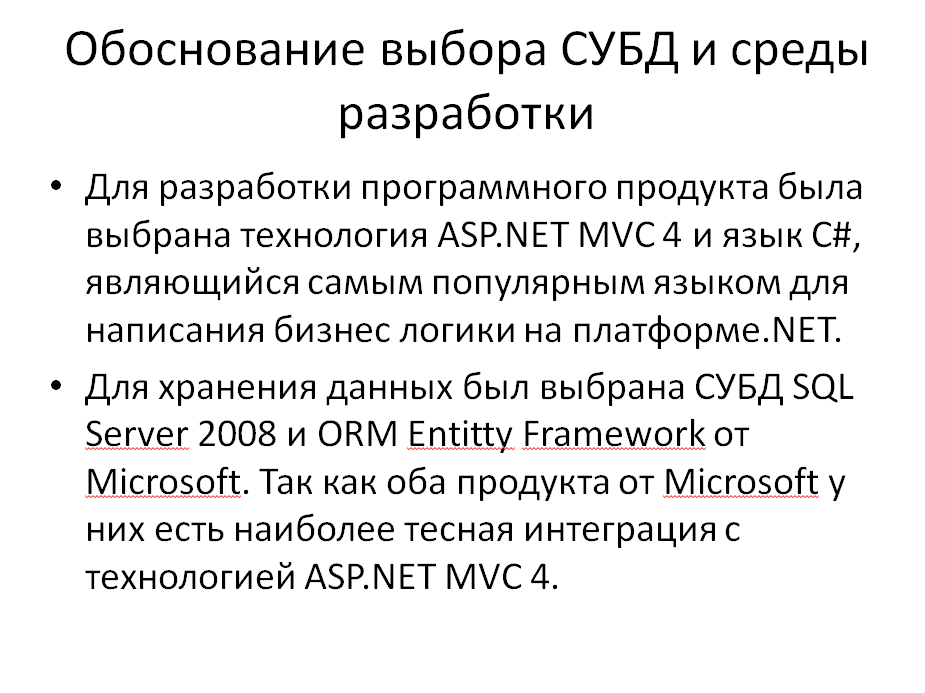
Слайд№7 – Концептуальное проектирование



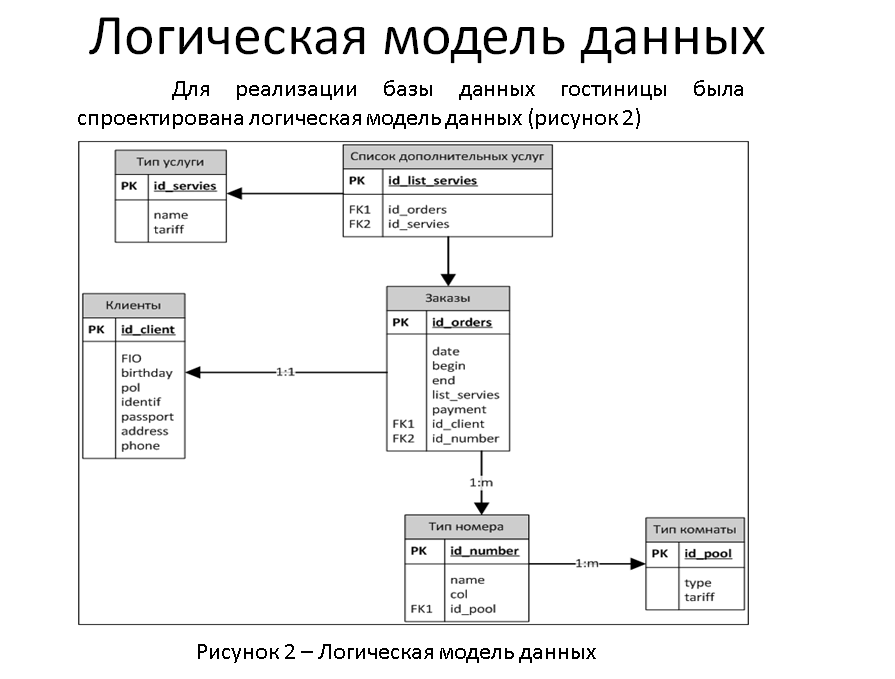
Слайд№8 – Концептуальное проектирование



Слайд№9 – Концептуальное проектирование



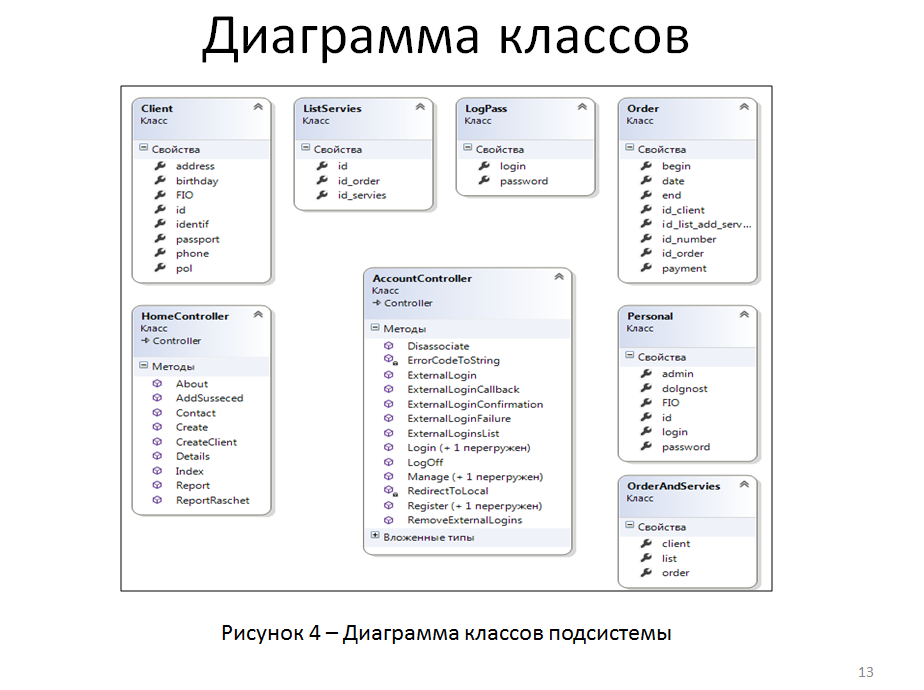
Слайд Г.10 – Обоснование выбора СУБД и инструментальные среды разработки



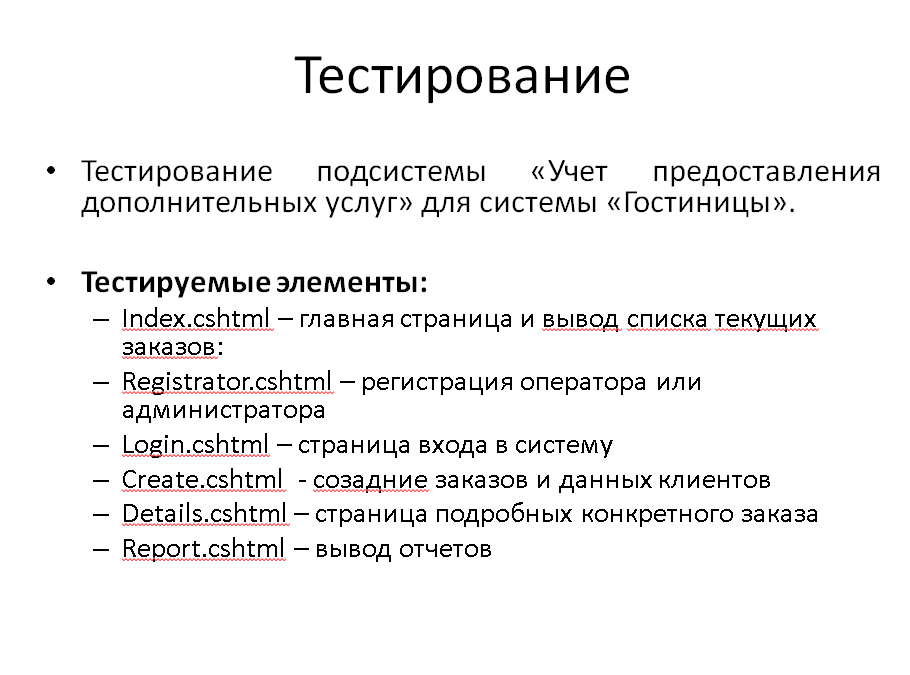
Слайд Г.11 – Логическая модель данных



Слайд Г.12 – Физическая модель данных



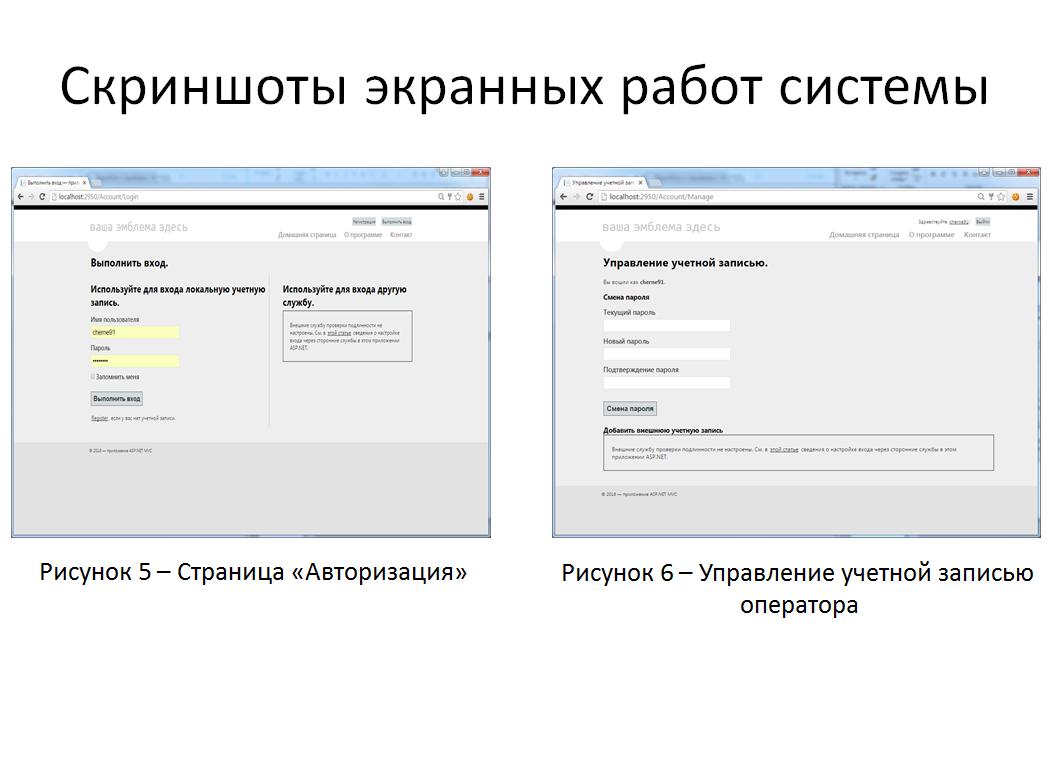
Слайд Г.13 – Диаграмма классов



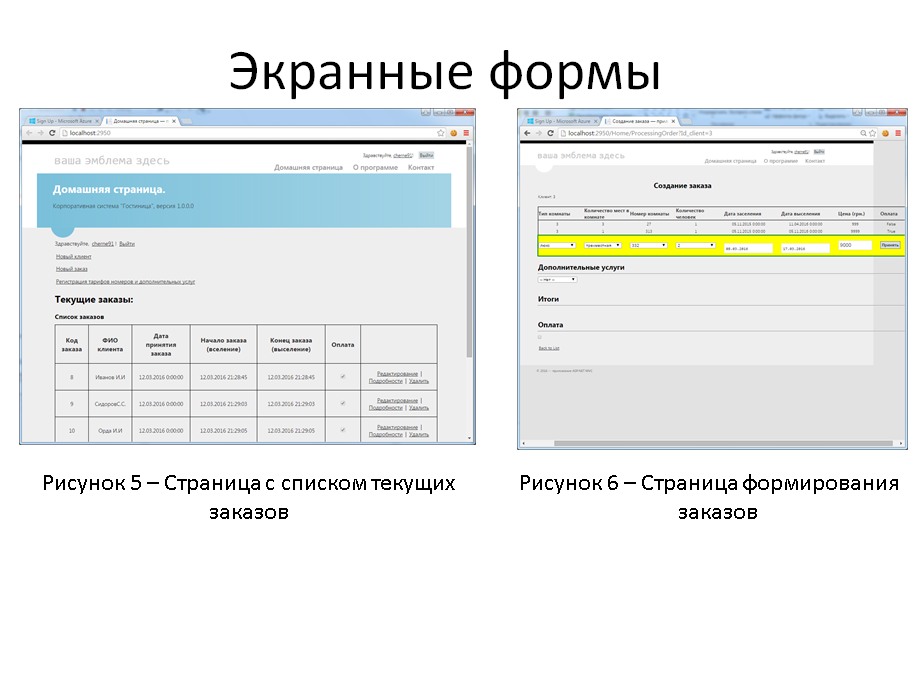
Слайд Г.14 – Тестирование и тестируемые элементы



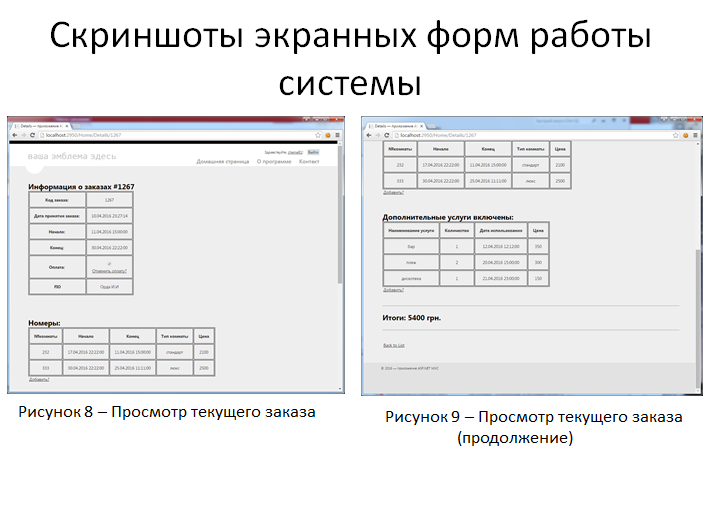
Слайд Г.15 – Таблица проведения тестирования с результатами



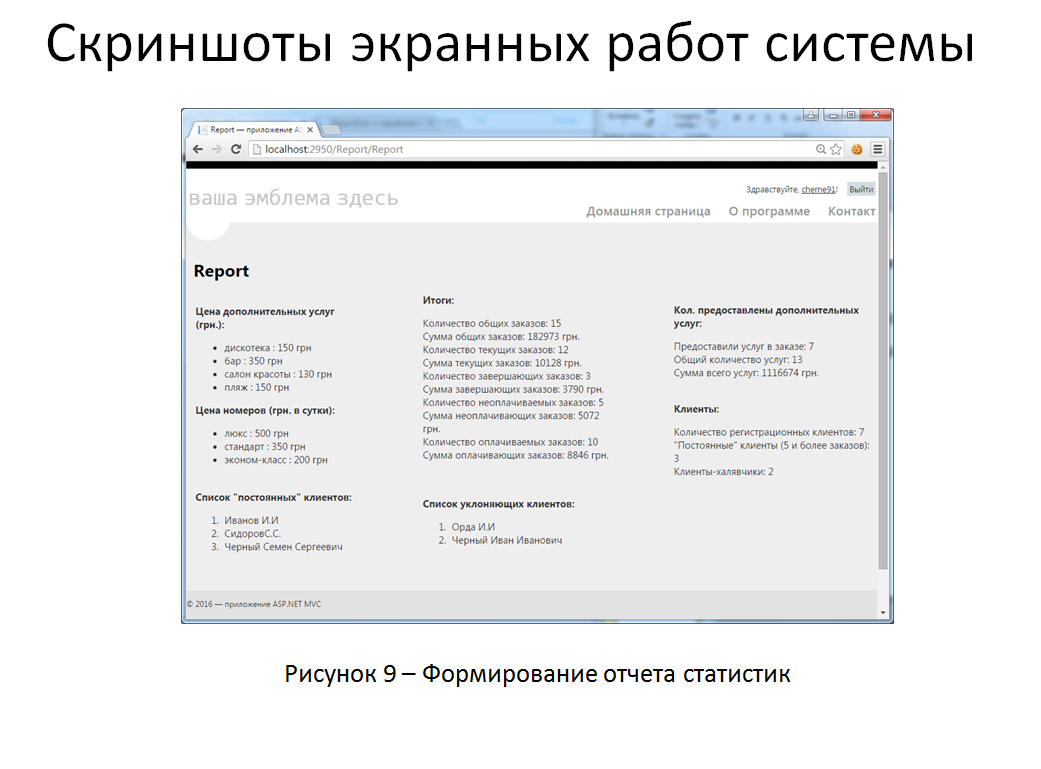
Слайд Г.16 – Скриншоты экранных работ системы: авторизация и страница «личного кабинета»



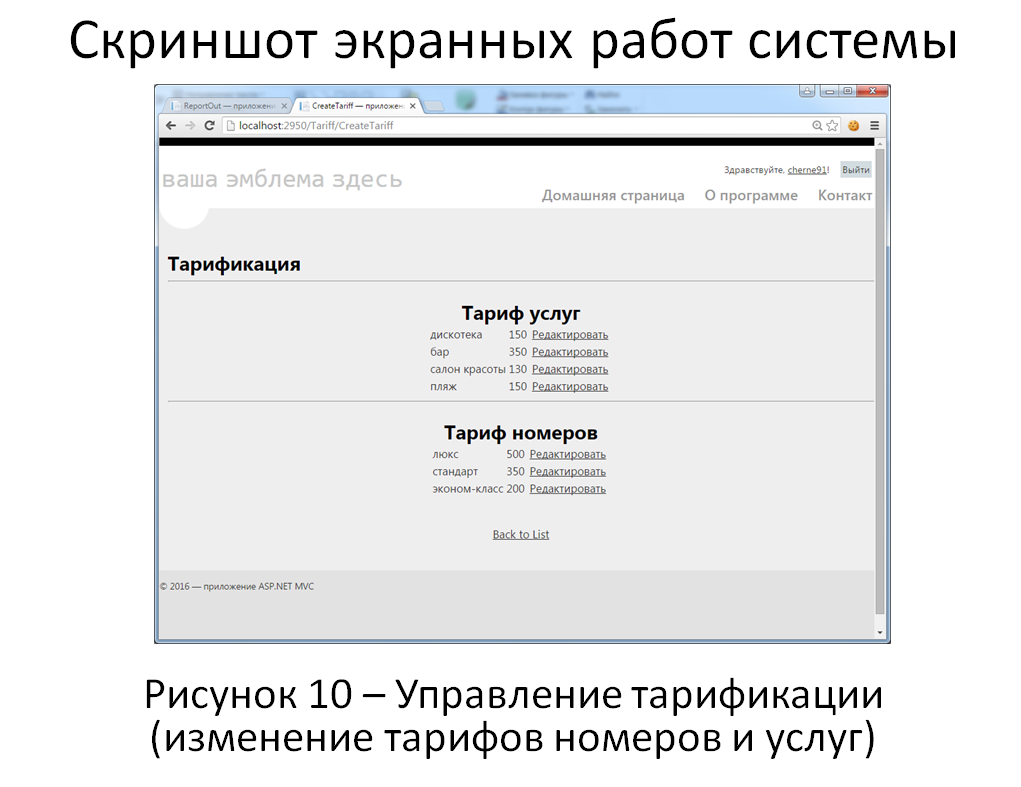
Слайд Г.17 – Скриншоты экранных работ системы: страница с текущими заказами и страница формирования заказов



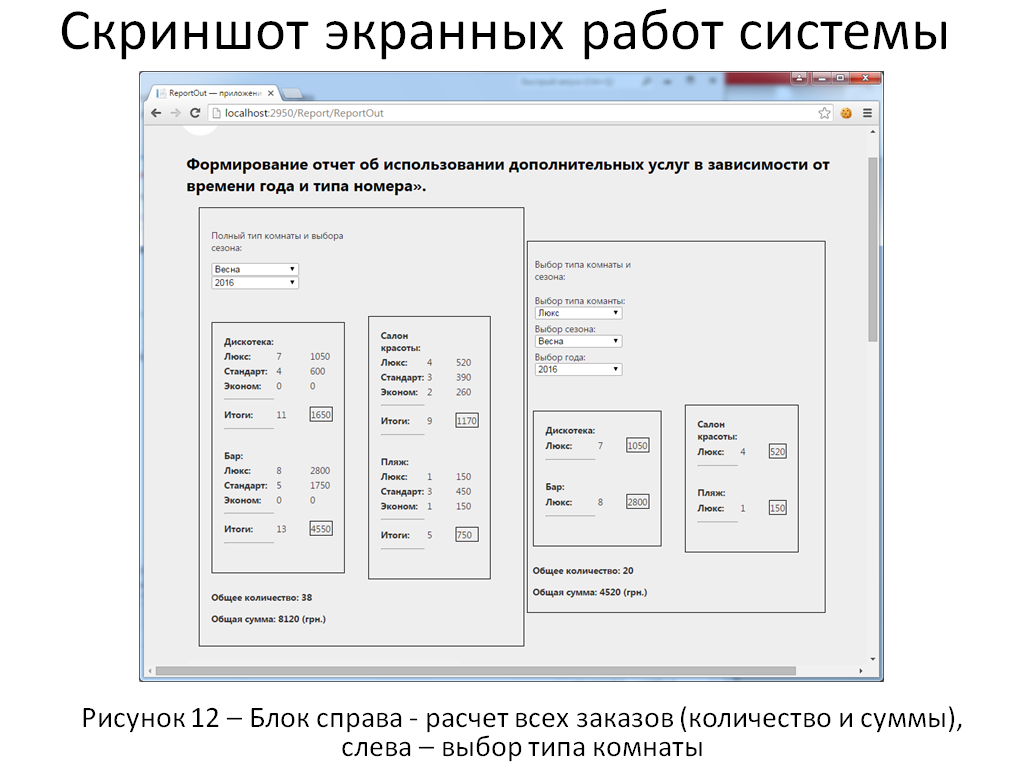
Слайд Г.18 – Просмотр текущего заказа



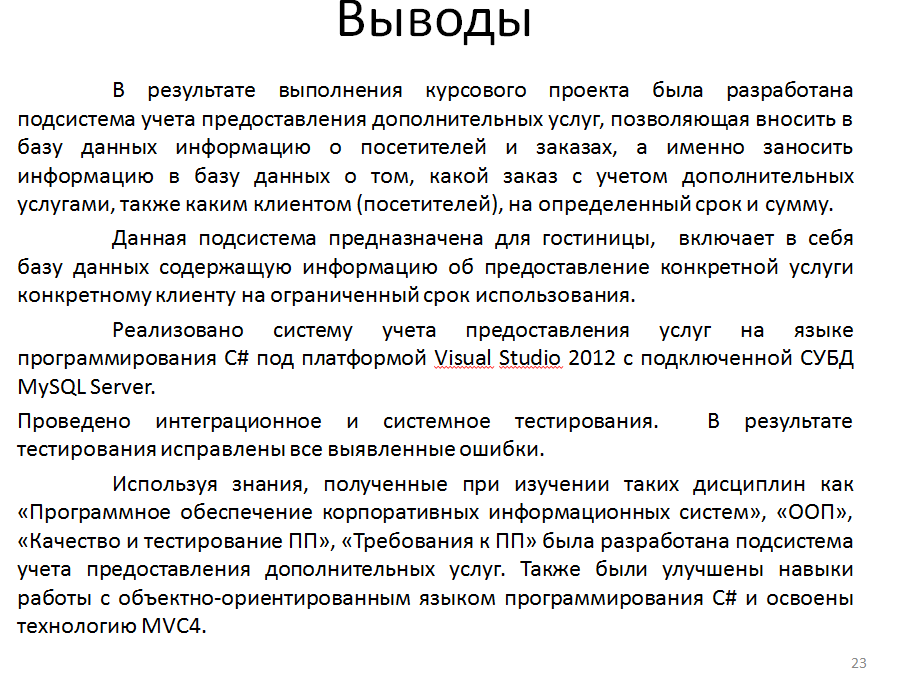
Слайд Г.19 – Формирование отчета статистик



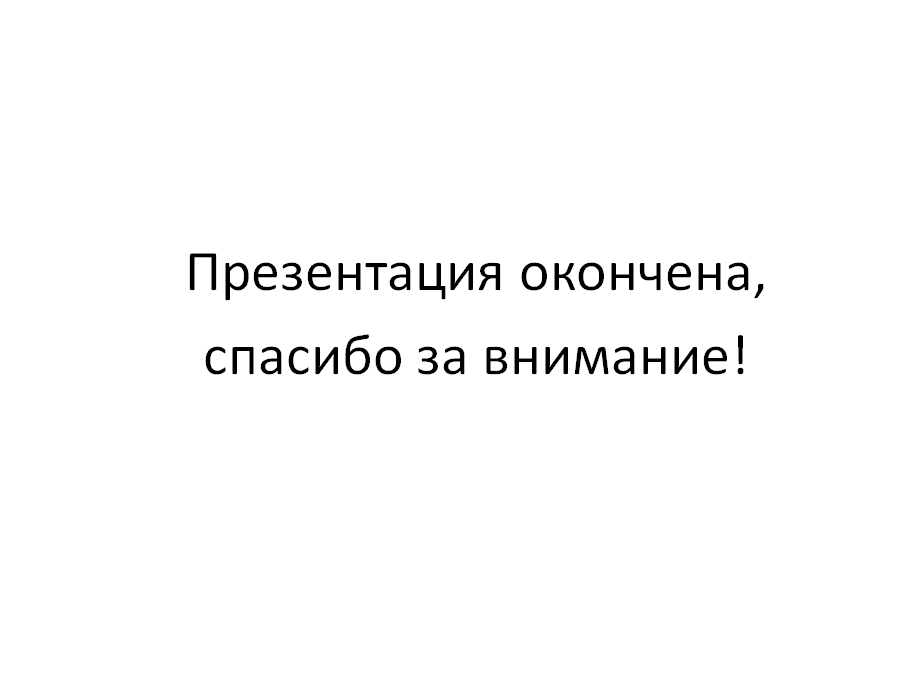
Слайд Г.20 – Формирование отчета



Слайд Г.21 – Расчет



Слайд Г.22 – Выводы



Слайд Г.23 – Надпись «Презентация окончена»