**Министерство образования Республики Беларусь**

**Учреждение образования**

**«Белорусский государственный университет информатики и радиоэлектроники»**

Факультет непрерывного и дистанционного обучения

Специальность «ИиТП»

**КОНТРОЛЬНАЯ РАБОТА № 2**

по предмету «Технологии разработки программного обеспечения»

по теме «Проектирование и разработка ПО на языке UML»

Студент дистанционной формы

обучения 5 курса

Группы № 493551

Авхимович Алексей Валерьевич

г.Борисов, ул. Чапаева, 29/116

Тел. +375-44-7900085

Минск 2019

Содержание

[1 Постановка задачи 4](#_Toc484602847)

[2 Анализ предметной области 5](#_Toc484602848)

[2.1 Разработка функциональной модели 5](#_Toc484602849)

[2.2 Разработка функциональных требований к системе 6](#_Toc484602850)

[3 Разработка базы данных 9](#_Toc484602851)

[4 Проектирование приложения 16](#_Toc484602852)

[4.1 Диаграммы классов 16](#_Toc484602853)

[4.2 Диаграмма состояний 18](#_Toc484602854)

[4.3 Диаграмма активности 20](#_Toc484602855)

[4.4 Диаграмма компонентов 21](#_Toc484602856)

[5 Реализация приложения 23](#_Toc484602857)

[Список использованных источников 25](#_Toc484602858)

[Приложение А Иллюстрации работы программного средства 26](#_Toc484602859)

[Приложение Б Фрагменты программного кода 31](#_Toc484602860)

# Постановка задачи

Предметная область: Гостиница. Отдел клиентского сервиса. Процесс бронирования и сдачи номеров.

Разработать программное средство для работы с базой данных.

Тип СУБД – Microsoft SQL Server.

Среда разработки – Microsoft Visual Studio 2017 Community.

# Анализ предметной области

## Разработка функциональной модели

Функциональную модель предметной области представим в виде диаграммы вариантов использования в нотации UML, представляющей систему в виде набора варианта использования и актеров, взаимодействующих с ними.

В рамках предметной области можно выделить двух актеров:

* администратор;
* менеджер.

Диаграмма вариантов использования системы приведена на рисунке 1.1.



Рисунок . – Диаграмма вариантов использования

Согласно приведенной диаграмме менеджер применяет следующие варианты использования:

* вести справочник типов номеров – просмотр, добавление, редактирование, удаление типов номеров;
* вести справочник гостиничных номеров – просмотр, добавление, редактирование, удаление номеров;
* регистрировать движения постояльцев – просмотр, добавление, редактирование, удаление движений;
* формировать отчетов – формирование отчетов по результатам деятельности.

Администратор применяет варианты использования, доступные менеджеру, а также дополнительный вариант использования «Вести справочник пользователей» – просмотр, добавление, редактирование и удаление данных пользователей.

Также на диаграмме приведен вспомогательный вариант использования «Авторизоваться», реализующий защищенный доступ к данным системы на основании парольного доступа.

## Разработка функциональных требований к системе

Выделенные варианты использования представляют собой основные функции системы.

К функции авторизации предъявляются следующие требования:

* для авторизации пользователь выбирает учетную запись из списка и вводит пароль;
* в случае неудачной авторизации необходимо вывести сообщение об ошибке.

К функции ведения справочника пользователей предъявляются следующие требования:

* все пользователи имеют одну из двух ролей: администратор или менеджер;
* доступ к справочнику пользователей имеет только пользователь с ролью администратор;
* администратор не может изменить свою роль;
* администратор не может удалить собственную запись;
* нельзя удалить пользователя, который участвовал в регистрации движений постояльцев.

К функции ведения справочника типов номеров предъявляются следующие требования:

* справочник ведется в соответствии с международной классификацией гостиничных номеров;
* для типа гостиничного номера указывают международное обозначение и пояснение на русском языке;
* нельзя удалить тип, которому поставлены в соответствие один или более гостиничных номеров.

К функции ведения справочника гостиничных номеров предъявляются следующие требования:

* цифровое обозначение гостиничного номера должно формироваться автоматически из номера этажа и порядкового номера комнаты на этаже (например, 412 – четвертый этаж, 12-я комната);
* для каждого гостиничного номера необходимо указать тип и максимально возможное количество постояльцев;
* нельзя удалить гостиничный номер, по которому были движения постояльцев.

К функции регистрации движения постояльцев предъявляются следующие требования:

* движения постояльцев бывают двух видов: бронирование и проживание;
* при регистрации движения необходимо указать дату регистрации события и пользователя, зарегистрировавшего движение;
* при регистрации необходимо указать гостиничный номер, дату заселения, количество дней проживания, дату выселения, а также количество проживающих.

К функции формирования отчетов предъявляются следующие требования:

* формируются следующие отчеты: количество проживающих на определенную дату, сведения о бронированиях номеров, сведения об имеющихся гостиничных номерах;
* отчет по количеству проживающих формируется в разрезе этажей и включает максимально доступное и фактическое количество проживающих на этаже, а также итоговые значения;
* отчет по бронированию формируется в разрезе движений бронирования и включает дату бронирования, номер комнаты, дату проживания, дату выселения, количество дней проживания, количество проживающих, а также итоговые количественные показатели;
* отчет по имеющимся гостиничным номерам формируется в разрезе типов гостиничных номеров и включает сведения о количестве номеров каждого типа, а также итоговый показатель.

# Разработка базы данных

Разработка базы данных включает разработку кода генерации таблиц, кода триггеров, кода представлений и хранимых процедур, а также кода запросов, которые будут применяться в приложении.

Разработку базы данных выполним с использованием СУБД Microsoft SQL Sever 2016 Express и среды управления базами данных Microsoft SQL Server Management Studio.

Рассмотрим код генерации таблиц базы данных в соответствии с разработанным в контрольной работе №1 проектом.

Код генерации таблицы Roles:

CREATE TABLE [dbo].[Roles] (

[Id] INT NOT NULL,

[Name] NVARCHAR (20) NOT NULL,

PRIMARY KEY CLUSTERED ([Id] ASC)

);

Код генерации таблицы Users:

CREATE TABLE [dbo].[Users] (

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

[Name] NVARCHAR (50) NOT NULL,

[Password] NVARCHAR (50) NOT NULL,

[RoleId] INT DEFAULT ((2)) NOT NULL,

PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_Users\_Roles] FOREIGN KEY ([RoleId]) REFERENCES [dbo].[Roles] ([Id])

);

Код генерации таблицы Types:

CREATE TABLE [dbo].[Types] (

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

[Code] NVARCHAR (20) NOT NULL,

[Name] NVARCHAR (50) NOT NULL,

PRIMARY KEY CLUSTERED ([Id] ASC)

);

Код генерации таблицы Actions:

CREATE TABLE [dbo].[Actions] (

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

[Name] NVARCHAR (20) NOT NULL,

PRIMARY KEY CLUSTERED ([Id] ASC)

);

Код генерации таблицы Rooms:

CREATE TABLE [dbo].[Rooms] (

[Id] INT NOT NULL,

[Floor] INT NOT NULL,

[Number] INT NOT NULL,

[TypeId] INT NOT NULL,

[PersonsMax] INT DEFAULT ((1)) NOT NULL,

PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_Rooms\_Types] FOREIGN KEY ([TypeId]) REFERENCES [dbo].[Types] ([Id])

);

К таблице Rooms добавим два триггера, отвечающие за формирование уникального кода – цифрового обозначения гостиничного номера по формуле:

Floor \* 100 + Number

где Floor – номер этажа;

Number – порядковый номер комнаты на этаже.

Первый триггер выполняется при вставке записи, заменяя собой соответствующую процедуру:

CREATE TRIGGER [dbo].[Trigger\_Rooms\_Insert\_For\_Id]

ON [dbo].[Rooms]

INSTEAD OF INSERT

AS

DECLARE @R\_Id int, @R\_Floor int, @R\_Number int, @R\_TypeId int, @R\_Persons int;

DECLARE InsCursor CURSOR FOR Select Floor, Number, TypeId, PersonsMax FROM Inserted;

BEGIN

OPEN InsCursor;

FETCH NEXT FROM InsCursor INTO @R\_Floor, @R\_Number, @R\_TypeId, @R\_Persons;

WHILE @@FETCH\_STATUS = 0

BEGIN

SET @R\_Id = @R\_Floor \* 100 + @R\_Number;

INSERT INTO Rooms (Id, Floor, Number, TypeId, PersonsMax) VALUES (@R\_Id, @R\_Floor, @R\_Number, @R\_TypeId, @R\_Persons)

FETCH NEXT FROM InsCursor INTO @R\_Floor, @R\_Number, @R\_TypeId, @R\_Persons;

END;

CLOSE InsCursor;

DEALLOCATE InsCursor;

END

Второй триггер выполняется после обновления записи:

CREATE TRIGGER [dbo].[Trigger\_Rooms\_Update\_For\_Id]

ON [dbo].[Rooms]

AFTER UPDATE

AS

DECLARE @R\_Floor int, @R\_Number int;

BEGIN

SELECT @R\_Floor = Floor, @R\_Number = Number FROM inserted

UPDATE Rooms SET Id = @R\_Floor \* 100 + @R\_Number

WHERE (Floor = @R\_Floor AND Number = @R\_Number)

END

Код генерации таблицы Moves:

CREATE TABLE [dbo].[Moves] (

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

[Dates] DATE NOT NULL,

[ActionId] INT NOT NULL,

[RoomId] INT NOT NULL,

[DatesIn] DATE NOT NULL,

[DaysCnt] INT NOT NULL,

[PersonsCnt] INT NOT NULL,

[UserId] INT NOT NULL,

[DatesOut] AS (dateadd(day,[DaysCnt],[DatesIn])),

PRIMARY KEY CLUSTERED ([Id] ASC),

CONSTRAINT [FK\_Moves\_Actions] FOREIGN KEY ([ActionId]) REFERENCES [dbo].[Actions] ([Id]),

CONSTRAINT [FK\_Moves\_Rooms] FOREIGN KEY ([RoomId]) REFERENCES [dbo].[Rooms] ([Id]),

CONSTRAINT [FK\_Moves\_Users] FOREIGN KEY ([UserId]) REFERENCES [dbo].[Users] ([Id])

);

Поле DatesOut таблицы Moves объявим как вычисляемое поле, значение которого определяется с помощью SQL-функции dateadd(), прибавляющей к дате определенное значение.

К таблице Moves добавим триггер, проверяющий значение поля PersonsCnt (количество постояльцев) и приводящий его в соответствие с максимально допустимым количеством постояльцев, указанным в таблице Rooms:

CREATE TRIGGER [dbo].[Trigger\_Moves\_PersonsCnt]

ON [dbo].[Moves]

AFTER INSERT, UPDATE

AS

DECLARE @P\_Max int, @P\_Current int, @M\_Id int, @M\_RoomId int;

BEGIN

Select @P\_Current = PersonsCnt, @M\_Id = Id, @M\_RoomId = RoomId FROM inserted;

Select @P\_Max = PersonsMax FROM Rooms WHERE Id = @M\_RoomId;

IF (@P\_Current > @P\_Max) UPDATE Moves SET PersonsCnt = @P\_Max WHERE Id = @M\_Id;

END

Триггер выполняется после операций вставки и обновления записей таблицы Moves.

Далее разработаем представления, включающие данные из нескольких таблиц, которые будем использовать при формировании табличных форм списков справочников и движений.

Представление UsersView включает данные пользователей, собираемые из таблиц Users и Roles. Данные объединяются с помощью внутреннего соединения (Inner Join):

CREATE VIEW [dbo].[UsersView]

AS SELECT Users.Id, Users.Name, Users.Password, Users.RoleId, Roles.Name AS RoleName

FROM Users INNER JOIN Roles ON Users.RoleId = Roles.Id

Представление RoomsView включает данные гостиничных номеров, собираемые из таблиц Rooms и Types. Данные объединяются с помощью внутреннего соединения (Inner Join):

CREATE VIEW [dbo].[RoomsView]

AS SELECT Rooms.Id, Rooms.Floor, Rooms.Number, Rooms.TypeId, Types.Code, Rooms.PersonsMax

FROM Rooms INNER JOIN Types ON Rooms.TypeId = Types.Id

Представление MovesView включает данные движения постояльцев, собираемые из таблиц Moves, Actions и Users. Данные объединяются с помощью внутреннего соединения (Inner Join):

CREATE VIEW [dbo].[MovesView]

AS SELECT Moves.Id, Moves.Dates, Moves.ActionId, Actions.Name AS ActionName, Moves.RoomId, Moves.DatesIn, Moves.DaysCnt, Moves.DatesOut, Moves.PersonsCnt, Moves.UserId, Users.Name AS UserName

FROM Moves INNER JOIN Actions ON Moves.ActionId = Actions.Id

INNER JOIN Users ON Moves.UserId = Users.Id

Далее разработаем хранимые процедуры, реализующие добавление, обновление и удаление записей для таблиц Moves и Rooms.

Хранимая процедура MoveDelete реализует удаление записи гостиничного номера:

CREATE PROCEDURE [dbo].MoveDelete

(

@Id int

)

AS

SET NOCOUNT OFF;

DELETE FROM Moves

WHERE (Id = @Id)

Хранимая процедура MoveInsert реализует добавление записи гостиничного номера:

CREATE PROCEDURE [dbo].MoveInsert

(

@Dates date,

@ActionId int,

@RoomId int,

@DatesIn date,

@DaysCnt int,

@PersonsCnt int,

@UserId int

)

AS

SET NOCOUNT OFF;

INSERT INTO Moves (Dates, ActionId, RoomId, DatesIn, DaysCnt, PersonsCnt, UserId)

VALUES (@Dates, @ActionId, @RoomId, @DatesIn, @DaysCnt, @PersonsCnt, @UserId)

Хранимая процедура MoveUpdate реализует обновление записи гостиничного номера:

CREATE PROCEDURE [dbo].MoveUpdate

(

@Dates date,

@ActionId int,

@RoomId int,

@DatesIn date,

@DaysCnt int,

@PersonsCnt int,

@Id int

)

AS

SET NOCOUNT OFF;

UPDATE Moves

SET Dates = @Dates, ActionId = @ActionId, RoomId = @RoomId, DatesIn = @DatesIn, DaysCnt = @DaysCnt, PersonsCnt = @PersonsCnt

WHERE (Id = @Id)

Аналогично реализуем хранимые процедуры для таблицы Rooms:

* RoomDelete – удаление записи гостиничного номера;
* RoomInsert – добавление записи гостиничного номера;
* RoomUpdate – обновление гостиничного номера.

Далее разработаем дополнительные запросы, которые будем применять в приложении для обращения к данным.

Для таблиц Roles, Actions и Types разработаем запросы на выборку всех данных.

Для таблиц Types и Users разработаем запросы на манипуляции с данными:

* GetById – выборка записи по идентификатору;
* InsertQuery – добавление записи;
* UpdateQuery – обновление записи;
* DeleteQuery – удаление записи.

Для выборки одиночных значений разработаем следующие запросы:

* CntMovesByRoom – возвращает количество движений по идентификатору гостиничного номера;
* CntMovesByUser – возвращает количество движений по идентификатору пользователя;
* CntRoomsByType – возвращает количество гостиничных номеров по идентификатору типа номера;
* RoleIdByUserData – возвращает код роли по идентификатору и паролю пользователя (если запись не найдена – возвращает NULL).

Далее разработаем запросы на выборку данных для формирования отчетов.

Запрос ReportPersonsCnt выбирает данные для отчета по количеству проживающих на указанную дату:

SELECT Rooms.Floor, SUM(Rooms.PersonsMax) AS SumMax, SUM(Moves.PersonsCnt) AS SumFact

FROM Moves INNER JOIN

Rooms ON Moves.RoomId = Rooms.Id

WHERE (Moves.DatesIn <= @Dates) AND (Moves.DatesOut >= @Dates) AND (Moves.ActionId = 2)

GROUP BY Rooms.Floor

Запрос ReportReservation выбирает данные для отчета по бронированию номеров:

SELECT Id, Dates, RoomId, DatesIn, DatesOut, DaysCnt, PersonsCnt

FROM Moves

WHERE (ActionId = 1) AND (DatesIn >= @DatesStart) AND (DatesIn <= @DatesEnd)

Запрос ReportRoomType выбирает данные для отчета по типам гостиничных номеров:

SELECT Types.Code, Types.Name, COUNT(Rooms.Id) AS RoomCnt

FROM Rooms INNER JOIN

Types ON Rooms.TypeId = Types.Id

GROUP BY Types.Code, Types.Name

# Проектирование приложения

## Диаграммы классов

Разработаем диаграммы классов приложения с разделением на пакеты.

Диаграмма пакетов классов приложения приведена на рисунке 4.1.



Рисунок . – Диаграмма пакетов приложения

На диаграмме представлены следующие классы и пакеты:

* Program – класс запускаемого приложения;
* AuthForm – класс формы авторизации;
* Forms – пакет классов экранных форм справочников и движений;
* Reports – пакет классов экранных форм отчетов;
* HotelDataSet – пакет классов предметной области.

Класс запускаемого приложения включает следующие атрибуты глобального доступа:

* roleId – идентификатор авторизованного пользователя;
* userId – идентификатор авторизованного пользователя.

Диаграмма классов пакета Forms приведена на рисунке 4.2.



Рисунок . – Классы пакета Forms

Пакет Forms включает следующие классы:

* MainForm – главная форма, включающая список движений и главное меню;
* MoveForm – форма данных движения;
* RoomsList – форма списка гостиничных номеров;
* RoomForm – форма данных гостиничного номера;
* TypesList – форма списка типов номеров;
* TypeForm – форма данных типа номеров;
* UsersList – форма списка пользователей;
* UserForm – форма данных пользователя.

Формы списков включают типичные методы:

* menuNew() – обработчик пункта меню «Добавить», открывает форму данных нового элемента;
* menuEdit() – обработчик пункта меню «Изменить», открывает форму данных редактирования элемента;
* menuDelte() – обработчик пункта меню «Удалить», обрабатывает удаление элемента;
* RefreshTable() – обновляет данные списка элементов из БД.

Формы элементов включают типичные методы:

* btnOk\_Click() – обработчик кнопки «Ok», обрабатывает сохранение данных элемента;
* btnCanel\_Click() – обработчик кнопки «Отмена», обрабатывает закрытие формы элемента.

Диаграмма классов пакета Reports приведена на рисунке 4.3.



Рисунок . – Диаграмма классов пакета Reports

На диаграмме представлены следующие классы форм отчетов:

* RPersonsForm – форма отчета по количеству проживающих;
* RRoomTypeForm – форма отчета по типам номеров;
* RReservationForm – форма отчета по бронированию номеров.

Классы модели данных разработаем непосредственно при реализации приложения.

## Диаграмма состояний

Далее разработаем диаграмму состояний приложения (рисунок 4.4).



Рисунок . – Диаграмма состояний приложения

После запуска приложения система принимает состояние «Выведена форма авторизации».

В случае отмены авторизации система завершает работу.

В случае, если авторизация пройдена, система переходит к состоянию «Выведена главная форма».

Дальнейшие состояния зависят от функции или операции, выбранной пользователем.

Если пользователь выбирает операцию удаления движения – происходит переход к состоянию «Выведен запрос подтверждения удаления движения» с возвратом при подтверждении либо отмене.

Если пользователь выбирает операцию добавления или изменения движения – происходит переход к состоянию «Выведена форма движения» с возвратом при подтверждении либо отмене.

Если пользователь выбрал справочник, система принимает состояние «Выведен список справочника». Далее – в зависимости от выбранной операции:

* если пользователь закрывает список – происходит возврат к состоянию «Выведена главная форма»;
* если пользователь выбирает операцию удаления – происходит переход к состоянию «Выведен запрос подтверждения удаления справочника» с возвратом при подтверждении либо отмене;
* если пользователь выбирает операцию добавления или изменения – происходит переход к состоянию «Выведена форма справочника» с возвратом при подтверждении либо отмене.

Если пользователь выбрал отчет, система принимает состояние «Выведена форма отчета» с возвратом при закрытии формы.

## Диаграмма активности

Разработаем диаграмму активности для варианта использования «Авторизоваться» (рисунок 4.5).



Рисунок . – Диаграмма активности варианта использования «Авторизоваться»

Активность включает следующие действия:

* система выводит форму авторизации;
* пользователь вводит данные авторизации (если пользователь отменяет авторизацию, активность завершается);
* если данные авторизации введены, система получает результат авторизации;
* если авторизация неудачна, система выводит сообщение о неудачной авторизации и возвращает пользователя к вводу данных авторизации;
* если авторизация успешна, система сохраняет данные пользователя в глобальном доступе и выводит главную форму, завершая активность.

## Диаграмма компонентов

Разработаем диаграмму компонентов системы (рисунок 4.6) этапа разработки.



Рисунок . – Диаграмма компонентов

На этапе разработки система включает следующие компоненты:

* Program – компонент запускаемого приложения;
* AuthForm – форма авторизации;
* Forms – пакетный компонент, представляющий формы приложения;
* Reports – пакетный компонент, представляющий формы отчетов;
* ReportTemplates – пакетный компонент, представляющий шаблоны отчетов;
* HotelDataSet – пакетный компонент, реализующий модель данных и обеспечивающий взаимодействие с базой данных;
* HotelDB – компонент базы данных.

# Реализация приложения

Выполним разработку проекта приложения с использованием среды разработки Microsoft Visual Studio 2017 Community.

Создадим проект приложения Windows-форм с рабочим названием Hotel.

Структура файлов проекта, разработанная в соответствии с классами, выявленными в ходе проектирования, приведена на рисунке 5.1.

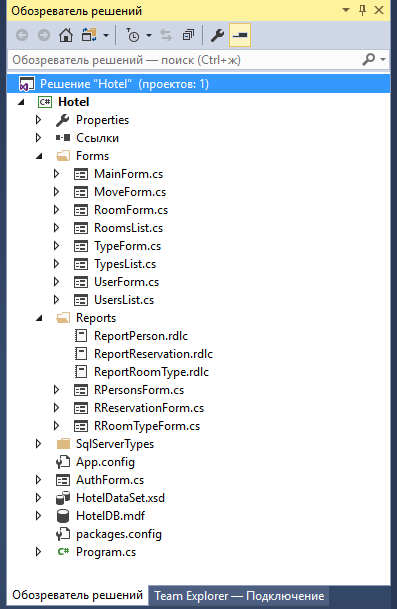


Рисунок 5.1 – Проект приложения

Схема модели данных HotelDataSet приведена на рисунке 5.2.

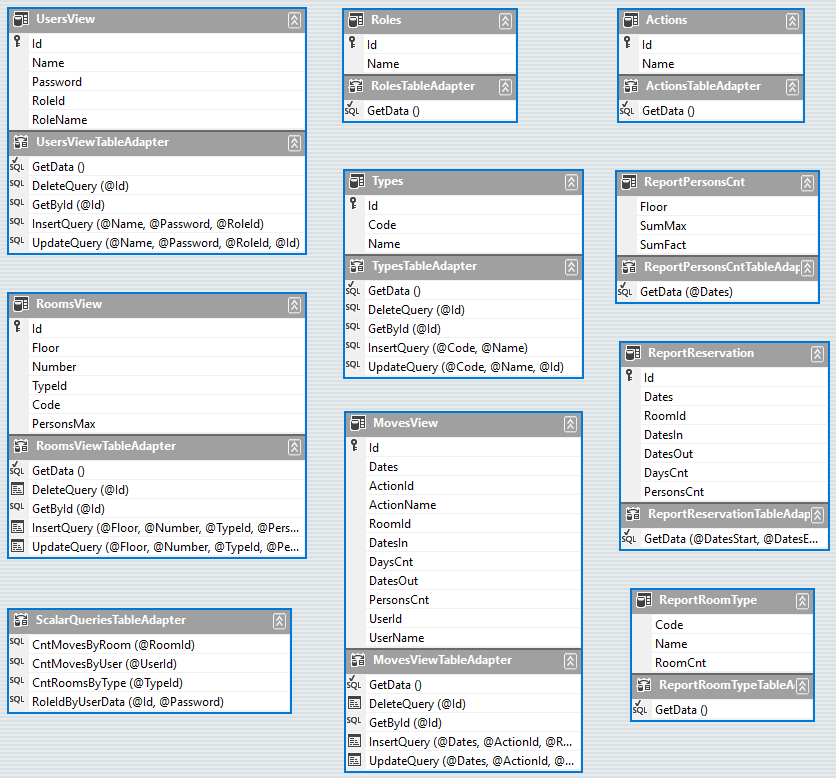


Рисунок 5.2 – Классы модели данных HotelDataSet

Приведенная схема включает классы, сформированные на основании таблиц, представлений, хранимых процедур и запросов базы данных, разработанной в разделе 3.

Для формирования отчетов к проекту с помощью менеджера пакетов NuGet добавлены следующие сборки:

* Microsoft.ReportingServices.ReportViewerControl.Winforms;
* Microsoft.SqlServer.Types.

Разработанные файлы отчета размещены в каталоге форм отчетов:

* ReportPerson.rdlc – отчет по количеству проживающих;
* ReportReservation.rdlc – отчет по бронированию номеров;
* ReportRoomType.rdlc – отчет по типам гостиничных номеров.

Экранные формы приложения приведены в приложении А, программный код приведен в приложении Б.

Список использованных источников

1. Леоненков, А. Самоучитель UML2 / А. Леоненков. – СПб.: БХВ-Петербург, 2014. – 576 с.
2. Уотсон, К. Visual C# 2010: базовый курс.: Пер. с англ. / Карли Уотсон [и др.]. – М.: ООО «Вильямс», 2010. – 1216 с.
3. Иллюстрации работы программного средства

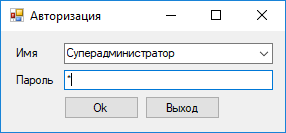


Рисунок А1 – Форма авторизации

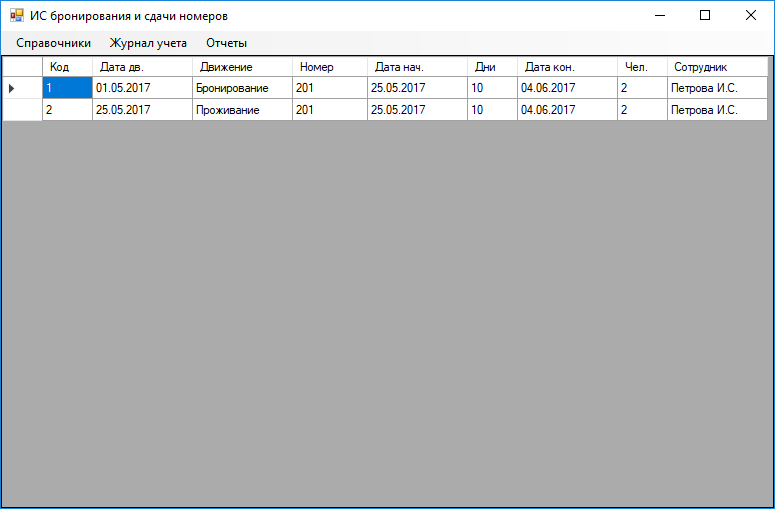


Рисунок А2 – Главная форма

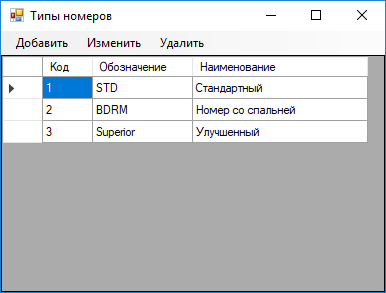


Рисунок А3 – Форма списка типов номеров

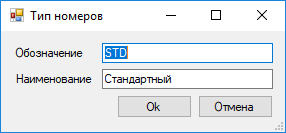


Рисунок А4 – Форма данных типа номеров

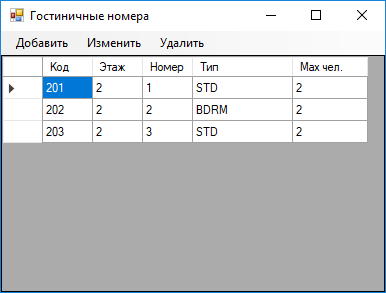


Рисунок А5 – Форма списка гостиничных номеров

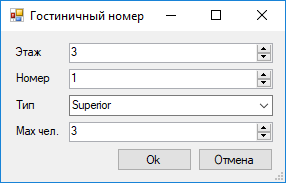


Рисунок А6 – Форма данных гостиничного номера

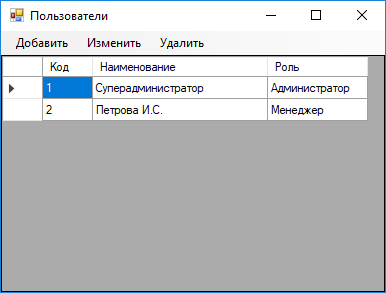


Рисунок А7 – Форма списка пользователей

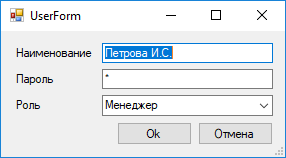


Рисунок А8 – Форма данных пользователя

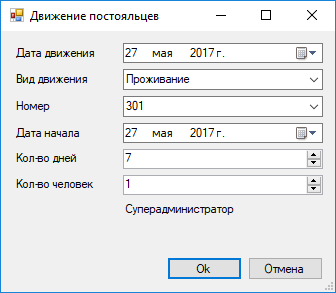


Рисунок А9 – Форма данных движения постояльцев

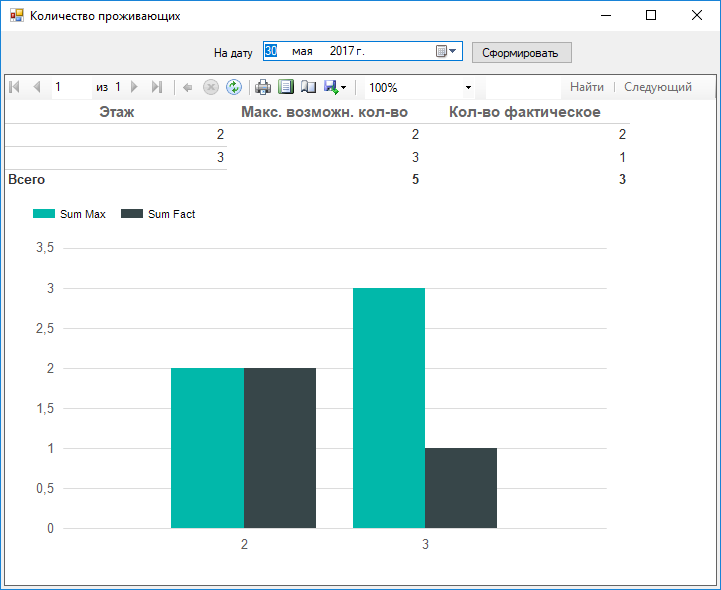


Рисунок А10 – Форма отчета по количеству проживающих

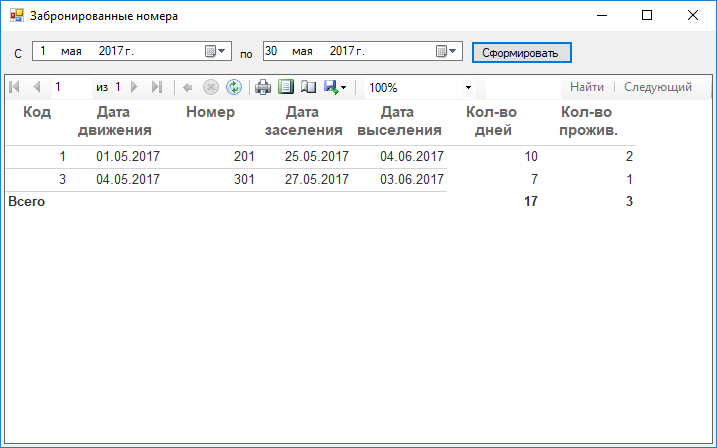


Рисунок А11 – Форма отчета по забронированным номерам

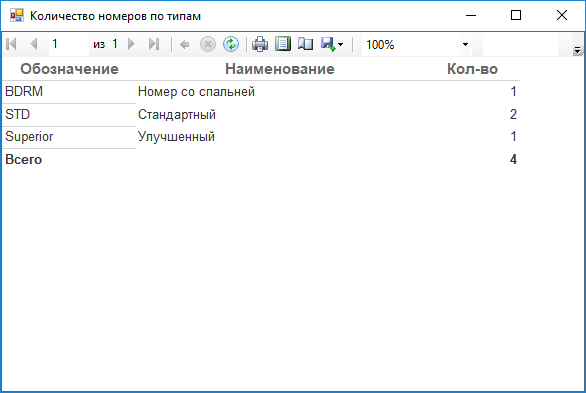


Рисунок А12 – Форма отчета по типам номеров

1. Фрагменты программного кода

AuthForm.cs

using System;

using System.Windows.Forms;

namespace Hotel

{

public partial class AuthForm : Form

{

public AuthForm()

{

InitializeComponent();

}

private void AuthForm\_Load(object sender, EventArgs e)

{

HotelDataSetTableAdapters.UsersViewTableAdapter ta = new HotelDataSetTableAdapters.UsersViewTableAdapter();

usersViewBindingSource.DataSource = ta.GetData();

}

private void btnOk\_Click(object sender, EventArgs e)

{

HotelDataSetTableAdapters.ScalarQueriesTableAdapter ta = new HotelDataSetTableAdapters.ScalarQueriesTableAdapter();

int userId = (int)fldName.SelectedValue;

int? roleId = ta.RoleIdByUserData(userId, fldPassword.Text);

if (roleId != null)

{

this.Hide();

Program.userId = userId;

Program.roleId = (int)roleId;

Forms.MainForm frm = new Forms.MainForm();

frm.ShowDialog();

this.Close();

}

else

{

MessageBox.Show(

"Авторизация неудачна",

"Внимание",

MessageBoxButtons.OK,

MessageBoxIcon.Warning

);

}

}

private void btnCancel\_Click(object sender, EventArgs e)

{

this.Close();

}

}

}

MainForm.cs

using System;

using System.Windows.Forms;

namespace Hotel.Forms

{

public partial class MainForm : Form

{

public MainForm()

{

InitializeComponent();

}

private void MainForm\_Load(object sender, EventArgs e)

{

RefreshTable();

if (Program.roleId != 1)

{

menuUsers.Visible = false;

}

}

private void RefreshTable()

{

HotelDataSetTableAdapters.MovesViewTableAdapter ta = new HotelDataSetTableAdapters.MovesViewTableAdapter();

movesViewBindingSource.DataSource = ta.GetData();

}

private void menuTypes\_Click(object sender, EventArgs e)

{

TypesList frm = new TypesList();

frm.ShowDialog();

}

private void menuRooms\_Click(object sender, EventArgs e)

{

RoomsList frm = new RoomsList();

frm.ShowDialog();

}

private void menuUsers\_Click(object sender, EventArgs e)

{

UsersList frm = new UsersList();

frm.ShowDialog();

}

private void menuMoveNew\_Click(object sender, EventArgs e)

{

MoveForm frm = new MoveForm(-1);

frm.ShowDialog();

RefreshTable();

}

private void menuMoveEdit\_Click(object sender, EventArgs e)

{

int itemId = (int)dgvMain["idDataGridViewTextBoxColumn", dgvMain.CurrentRow.Index].Value;

MoveForm frm = new MoveForm(itemId);

frm.ShowDialog();

RefreshTable();

}

private void menuMoveDel\_Click(object sender, EventArgs e)

{

int itemId = (int)dgvMain["idDataGridViewTextBoxColumn", dgvMain.CurrentRow.Index].Value;

if (MessageBox.Show(

"Вы действительно хотите удалить элемент?",

"Подтверждение удаления",

MessageBoxButtons.YesNo,

MessageBoxIcon.Warning,

MessageBoxDefaultButton.Button2

) == DialogResult.Yes)

{

HotelDataSetTableAdapters.MovesViewTableAdapter ta = new HotelDataSetTableAdapters.MovesViewTableAdapter();

ta.DeleteQuery(itemId);

RefreshTable();

}

}

private void menuReportPersonCnt\_Click(object sender, EventArgs e)

{

Reports.RPersonsForm frm = new Reports.RPersonsForm();

frm.ShowDialog();

}

private void menuReportReservation\_Click(object sender, EventArgs e)

{

Reports.RReservationForm frm = new Reports.RReservationForm();

frm.ShowDialog();

}

private void menuReportRoomType\_Click(object sender, EventArgs e)

{

Reports.RRoomTypeForm frm = new Reports.RRoomTypeForm();

frm.ShowDialog();

}

}

}

MoveForm.cs

using System;

using System.Windows.Forms;

namespace Hotel.Forms

{

public partial class MoveForm : Form

{

int itemId = -1;

public MoveForm(int itemId)

{

this.itemId = itemId;

InitializeComponent();

}

private void MoveForm\_Load(object sender, EventArgs e)

{

HotelDataSetTableAdapters.ActionsTableAdapter taAct = new HotelDataSetTableAdapters.ActionsTableAdapter();

fldAction.DataSource = taAct.GetData();

HotelDataSetTableAdapters.RoomsViewTableAdapter taRoom = new HotelDataSetTableAdapters.RoomsViewTableAdapter();

fldRoom.DataSource = taRoom.GetData();

HotelDataSetTableAdapters.MovesViewTableAdapter ta = new HotelDataSetTableAdapters.MovesViewTableAdapter();

bindingItem.DataSource = ta.GetById(itemId);

}

private void btnOk\_Click(object sender, EventArgs e)

{

HotelDataSetTableAdapters.MovesViewTableAdapter ta = new HotelDataSetTableAdapters.MovesViewTableAdapter();

if (itemId < 0)

{

ta.InsertQuery(fldDates.Value, (int)fldAction.SelectedValue, (int)fldRoom.SelectedValue, fldDatesIn.Value, (int)fldDays.Value, (int)fldPersons.Value, Program.userId);

}

else

{

ta.UpdateQuery(fldDates.Value, (int)fldAction.SelectedValue, (int)fldRoom.SelectedValue, fldDatesIn.Value, (int)fldDays.Value, (int)fldPersons.Value, itemId);

}

Close();

}

private void btnCancel\_Click(object sender, EventArgs e)

{

Close();

}

}

}