#### Министерство образования и науки Российской Федерации

#### Федеральное государственное бюджетное образовательное учреждение

#### высшего образования

#### «Владимирский государственный университет

#### имени Александра Григорьевича и Николая Григорьевича Столетовых»

Кафедра информационных систем и программной инженерии

**Этапы курсового проекта**

по дисциплине "Технологии программирования"

Выполнила:

ст. группы ПРИ-114

Глушнева Д.Д.

Принял:

Вершинин В.В.

г. Владимир, 2016 г.

**Этап курсового проекта №1**

**Описание предметной области**

Предметной областью данного курсового проекта является агентство по изготовлению наружной рекламы. В услуги агентства по изготовлению наружной рекламы входит:

* изготовление различных видов наружной рекламы (штендеры, вывески, информационные стенды, банерная печать и др.)
* изготовление некоторых видов полиграфии (визитки, буклеты, календари и др.)
* размещение и установка наружной рекламы (например, с помощью металлоконструкций)

Работа с клиентами проходит по следующей схеме: встреча с клиентом, выявление его пожеланий; изготовление макета изделия в программных средствах Corel Draw и Adobe Photoshop; в случае одобрения клиентом макета изготовление изделия и при надобности его монтаж. Оплата работ производится наличным и безналичным способом по желанию клиента, при этом предоплата составляет пятьдесят процентов и вносится на этапе одобрения макета. Оставшиеся пятьдесят процентов от суммы оплаты вносят при монтаже изделия.

Учредитель агентства по изготовлению наружной рекламы является его директором. Изготовление изделий и их монтаж производится сотрудниками агентства (работниками). Составлением, обработкой и ведением заказа занимаются менеджеры. Составлением макета изделия занимается дизайнер.

Для обеспечения автоматизации работы рекламного агентства в программной системе могут работать: директор агентства, менеджеры агентства, администратор, клиенты и неавторизованные пользователи системы.

В функциональные обязанности директора входит:

1. Договоры с поставщиками, доставка, оплата материалов.
2. Определение ценовых тарифов на изготавливаемые изделия (цена устанавливается на основе стоимости сырья с учетом затрат на электроэнергию, работу мастера и др.)
3. Ответственность за работу сотрудников.
4. Проведение указаний для работы сотрудников.
5. Выдача з/п сотрудникам.
6. Принятие человека на работу. Увольнение сотрудника с работы

В функциональные обязанности менеджера входит:

1. Ведение заказа клиента (Составление, принятие и ведение заказа).
2. Просмотр каталога изделий, акций, новостей, заказов.
3. Проверка оплаты заказа.

В функциональные обязанности администратора входит:

1. Составление и разработка акций на изделия.
2. Добавление в каталог новых изделий, или их удаление из каталога
3. Ведение каталога клиентов

**Моделирование структуры объектов предметной области и их взаимодействия на концептуальном уровне. Диаграмма прецедентов**

***Расширенное описание прецедента «Регистрация»***

**Название:** «Регистрация»

**Действующее лицо:** неавторизованный пользователь

**Основной поток**: *Регистрация*

Неавторизованный пользователь системы проходит процедуру регистрации в системе. Для этого он заполняет форму регистрации необходимыми данными (ФИО, e-mail, пароль). Система сохраняет информацию о пользователе в БД. После завершения регистрации в систему добавляется новый пользователь.

***Расширенное описание прецедента «Авторизация»***

**Название:** «Авторизация»

**Предусловие:** пользователь зарегистрирован в ИС "Рекламное агентство".

**Действующее лицо:** неавторизованный пользователь

**Основной поток**: *Авторизоваться*

Неавторизованный пользователь системы открывает форму (окно приложения), отображающую поля для ввода необходимой информации для авторизации в системе.

После ввода необходимой информации пользователь нажимает на кнопку "Авторизоваться". Введенная информация проверяется на наличие в БД. Если введенная информация корректна, то пользователь авторизуется в системе с назначенной ему ролью (клиент, менеджер, директор, администратор).

**Альтернативный поток**: Если введенная информация не корректна, то пользователю выводится сообщение об ошибки авторизации.

***Расширенное описание прецедента «Подача резюме»***

**Название:** «Подача резюме»

**Действующее лицо:** неавторизованный пользователь

**Основной поток**: *Подать резюме*

Неавторизованный пользователь системы открывает форму (окно приложения), отображающую поля для заполнения необходимой информации. После заполнения формы пользователь нажимает кнопку "Отправить резюме". Информация из резюме заносится в БД.

***Расширенное описание прецедента «Просмотр резюме»***

**Название:** «Просмотр резюме»

**Действующее лицо:** директор

**Основной поток**: *Просмотр*

Пользователь открывает форму (окно приложения), отображающее все резюме в БД системы. Пользователь может просматривать резюме, принять резюме или удалить резюме из базы. Система сохраняет состояние резюме в БД.

***Расширенное описание прецедента «Просмотр списка пользователей»***

**Название:** «Просмотр списка пользователей»

**Действующее лицо:** администратор

**Основной поток**: *Просмотр*

Администратор открывает форму (окно приложения), отображающую список пользователей системы. Администратор может создавать нового пользователя (менеджер ) , изменять данные пользователя (менеджер, директор, клиент (при изменении необходимых данных)), удалять пользователей. Система сохраняет состояние пользователя в БД.

***Расширенное описание прецедента «Просмотр новостей»***

**Название:** «Просмотр новостей»

**Предусловие:** Дополнением "Удалить новость" может пользоваться только администратор. Дополнением "Добавить отзыв" может пользоваться только клиент.

**Действующее лицо:** любой пользователь

**Основной поток**: *Просмотр*

Пользователь открывает форму (окно приложения), отображающую список новостей в системе. Администратор может удалять новость из БД (Пример: удалить за нецензурные выражения). Клиент может добавлять отзыв о работе рекламного агентства на фору. Система сохраняет состояние новостей в БД.

***Расширенное описание прецедента «Просмотр акций»***

**Название:** «Просмотр акций»

**Предусловие:** Дополнениями "Удалить акцию" и "Добавить акцию"может пользоваться только администратор.

**Действующее лицо:** любой пользователь

**Основной поток**: *Просмотр*

Пользователь открывает форму (окно приложения), отображающую список действующих акций на рекламу. Администратор может добавлять новую акцию и удалять акцию по прошествии времени. Система сохраняет состояние акций в БД.

***Расширенное описание прецедента «Просмотр изделий»***

**Название:** «Просмотр изделий»

**Предусловие:** Дополнениями "Удалить изделие" и "Добавить изделие"может пользоваться только администратор.

**Действующее лицо:** любой пользователь

**Основной поток**: *Просмотр*

Пользователь открывает форму (окно приложения), отображающую список изготавливаемых изделий. Администратор может добавлять новое изделие и удалять изделие, которое больше не изготавливают. Система сохраняет состояние изделий в БД.

***Расширенное описание прецедента «Просмотр заказа»***

**Название:** «Просмотр заказа»

**Предусловие:** Дополнениями "Составить заказ" и "Обработать заказ"может пользоваться только менеджер. Дополнением "Создать заказ" может пользоваться только клиент.

**Действующее лицо:** менеджер, клиент

**Основной поток**: *Просмотр*

Пользователь открывает форму (окно приложения), отображающую список выполняемых и требующих обработки заказов (для менеджера), отображающую список созданных заказов (для клиента). При создании клиент оплачивает заказ безналичными средствами оплаты, нажав на кнопку "Оплатить заказ". Менеджер обрабатывает электронный вариант заказа, который отправил клиент, или составляет его, если клиент пришел лично на предприятие. Система сохраняет состояние заказов в БД.

***Расширенное описание прецедента «Начисление заработной платы»***

**Название:** «Начисление заработной платы»

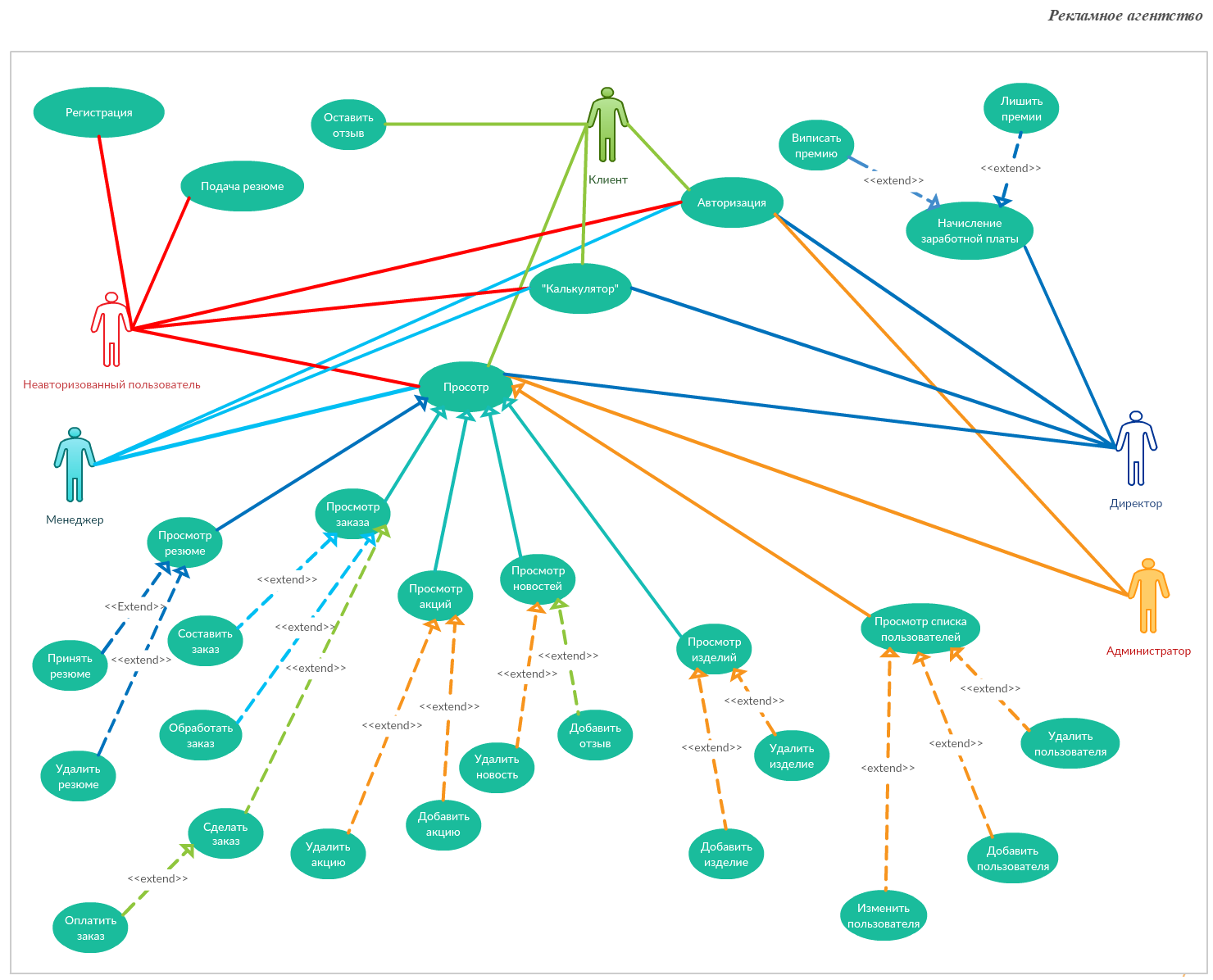
**Действующее лицо:** директор

**Основной поток**: *начисление заработной платы*

При начислении заработной платы сотрудникам рекламного агентства директор может выписать премию сотруднику или лишить премии. Система сохраняет состояние заработной платы в БД.

**Диаграмма прецедентов (В процессе доработки)**

Диаграмма прецедентов представлена на рисунке 1.



*Рисунок 1. Диаграмма прецедентов*

**Этап курсового проекта №2**

**Диаграмма классов и объектов (В процессе доработки)**

1. **Описание сущностей**

В ходе начального анализа было выделены следующие сущности:

* Пользователь;
* Заказ;
* Акция;
* Изделие;
* Новость;
* Резюме.

Перечень функций системы:

* Регистрация;
* Авторизация;
* Установка роли;
* Просмотр изделия;
* Просмотр акций;
* Просмотр новостей;
* Подача резюме;

Сущность «Пользователь» имеет следующие атрибуты:

* ФИО;
* Название роли.

Сущность «Резюме» имеет следующие атрибуты:

* Код резюме;
* ФИО (относится к роли "Неавторизованный клиент");
* Данные паспорта;
* Стаж работы (если «да», то где).

Сущность «Акция» имеет следующие атрибуты:

* Название акции;
* Срок проведения акции:
* Описание условий акции;

Сущность «Изделие» имеет следующие атрибуты:

* Код изделия;
* Название изделия:
* Описание изделия;
* Цена изделия.

Сущность «Заказ» имеет следующие атрибуты:

* Код заказа;
* ФИО заказчика;
* Наименование изделий;
* Количество изделий;
* Сумма заказа.

Сущность «Новость» имеет следующие атрибуты:

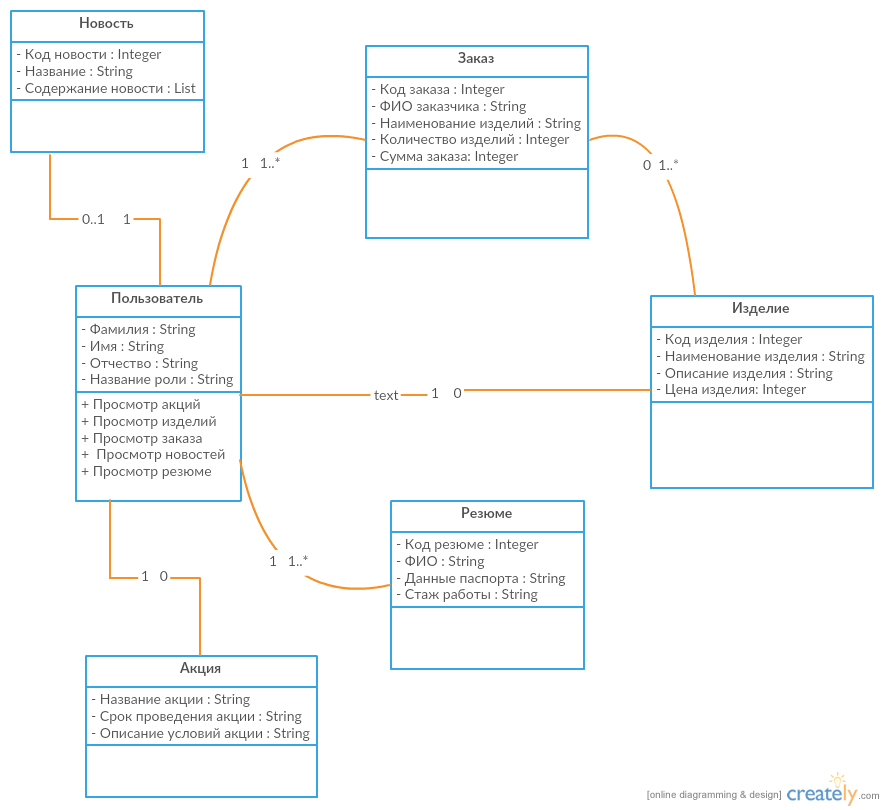
* Код новости;
* Название;
* Содержание новости.

1. **Описание связей между классами.**

* Между классами «Пользователь» и «Новость» стоит связь композиция 0..\* к 1. Так как пользователь может создать несколько новостей (отзывов) о работе агентства.
* Между классами «Пользователь» и «Резюме» стоит связь композиция 1 к \*. Так как у одного человека может быть несколько резюме.
* Между классами «Пользователь» и «Заказ» стоит связь композиция 1 к \*. Так как у человека может несколько заказов.
* Между классами «Заказ» и «Изделие» стоит связь композиция 0 к \*. Так как в заказе может быть несколько изделий.
* Между классами «Пользователь» и «Изделие» стоит связь композиция 0 к 1. Так как у человека может просматривать только одно изделие из списка.
* Между классами «Пользователь» и «Акция» стоит связь композиция 0 к 1. Так как у человека может просматривать только одно изделие из списка.

1. **Диаграмма классов**

Диаграмма класса представлена на рисунке 2.



*Рисунок 2. Диаграмма классов*

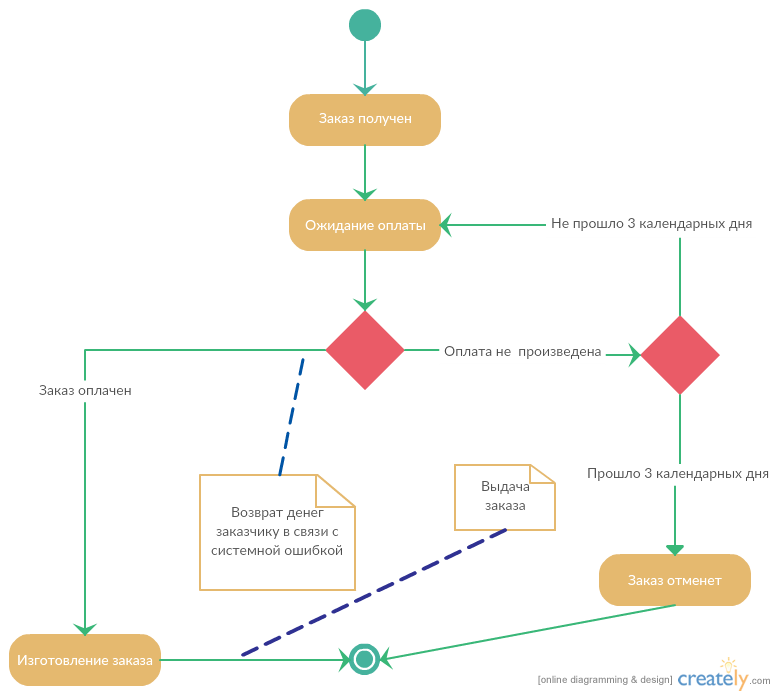
**Диаграмма состояний и последовательностей объектов**

1. **Объект " Заказ "**
   1. **Диаграмма состояния (В процессе доработки)**

Составляю диаграмму состояний объекта «Заказ» в ИС «Рекламное агентство». Один из возможных бизнес-процессов работы ИС с заказом представлен ниже:

1. Пользователь (Клиент) оформил заказ, указав свои реквизиты, способ оплаты. Объект «Заказ» создан в системе (запись в БД). Поскольку оплата заказа производится через внешнюю платежную систему, заказ не обрабатывается до момента получения подтверждения об оплате.
2. От платежной системы пришло сообщение об отказе в оплате или истек максимальный срок ожидания оплаты. Покупателю на указанный им e-mail отправляется соответствующее уведомление, после чего переходим к п. 6.
3. От платежной системы пришло подтверждение об оплате. Объект «Заказ» помечается как оплаченный. Менеджер агентства обрабатывает заказ и передает его в цех по производству изделий (вне системы).
4. Менеджер отмечает объект "Заказ" как выполненный после сообщения персонала о его готовности (вне системы) .
5. От системы пришло подтверждение о готовности заказа. Заказ переходит в состояние ожидания подтверждения о доставке. Покупателю отправляется уведомление о готовности заказа.
6. Заказ помечается как отмененный.

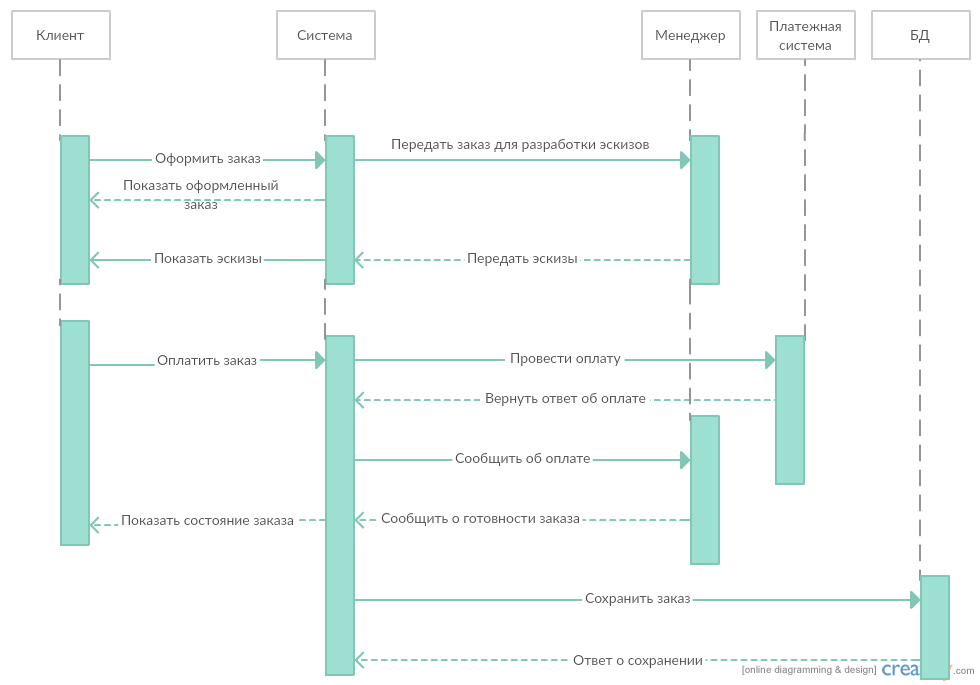
Пример диаграммы состояний для объекта «Заказ» представлен на рисунке 3.



*Рисунок 3. Диаграмма состояния объекта "Заказ"*

* 1. **Диаграмма последовательности**

Диаграмма последовательностей представлена на рисунке 4.



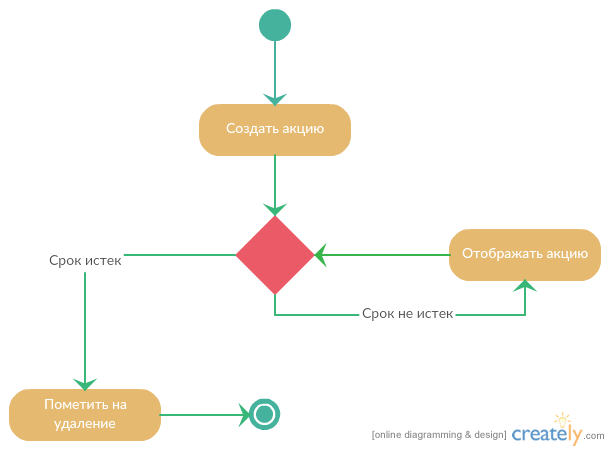
*Рисунок 4. Диаграмма последовательности*

1. **Объект " Акция "**
   1. **Диаграмма состояния**

Составляю диаграмму состояний объекта «Акция» в ИС «Рекламное агентство». Один из возможных бизнес-процессов работы ИС с акцией представлен ниже:

1. Пользователь (Администратор) создает акцию, указав ее название, срок проведения и описание условий акции. Объект «Акция» создан в системе (запись в БД). Акция действительна, пока не истек срок ее проведения
2. По истечении срока проведения, объект "Акция" помечается на удаление, после чего переходим к п. 3.
3. Системе пришло сообщение о истечении срока проведении акции. Объект «Акция» помечается на удаление и посылается команда в БД на удаление проведенной акции.

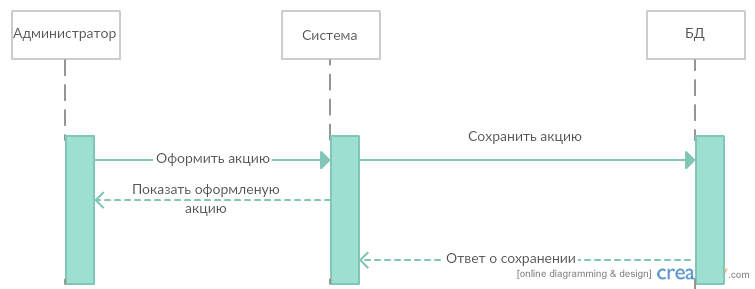
Пример диаграммы состояний для объекта «Акция» представлен на рисунке 5.



*Рисунок 5. Диаграмма состояния объекта "Акция"*

* 1. **Диаграмма последовательности**

Диаграмма последовательностей представлена на рисунке 6.



*Рисунок 6. Диаграмма последовательности*

**Этап курсового проекта №3**

1) Создана локальная база данных, содержащая небольшое количество данных, необходимых для просмотра и тестирования результатов функционала системы. По созданной базе данных созданы модели таблиц.

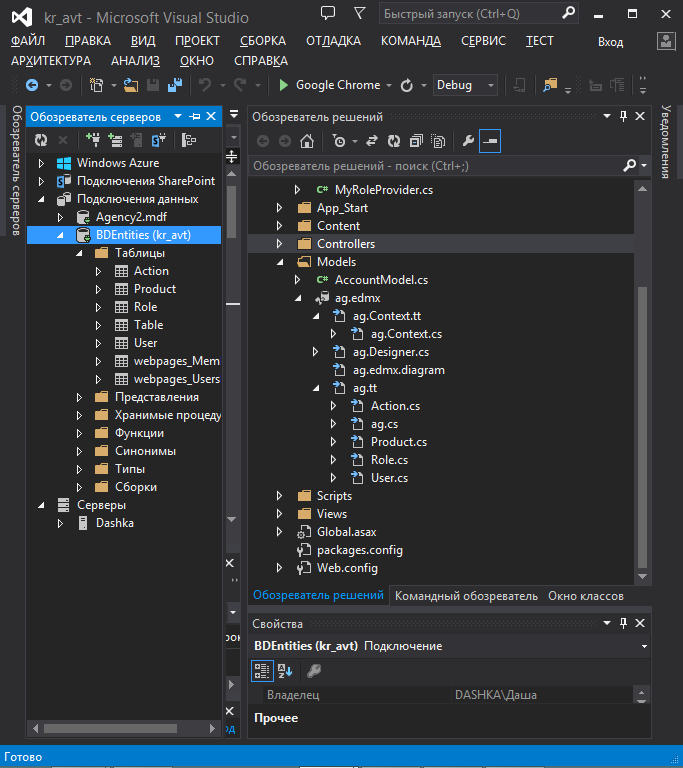


Рисунок 7

2) Cоздаю к каждой модели конроллер, организующий основную работу с данными модели (На данном этапе разработки созданы контроллеры UserController и ProductController). Пример кода UserController представлен ниже:

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using kr\_avt.Models;

namespace kr\_avt.Controllers

{

public class UserController : Controller

{

private BDEntities db = new BDEntities();

//

// GET: /User/

public ActionResult Index()

{

var user = db.User.Include(u => u.Role);

return View(user.ToList());

}

//

// GET: /User/Details/5

public ActionResult Details(int id = 0)

{

User user = db.User.Find(id);

if (user == null)

{

return HttpNotFound();

}

return View(user);

}

//

// GET: /User/Create

public ActionResult Create()

{

ViewBag.IDRole = new SelectList(db.Role, "IDRole", "NameRole");

return View();

}

//

// POST: /User/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(User user)

{

if (ModelState.IsValid)

{

db.User.Add(user);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.IDRole = new SelectList(db.Role, "IDRole", "NameRole", user.IDRole);

return View(user);

}

//

// GET: /User/Edit/5

public ActionResult Edit(int id = 0)

{

User user = db.User.Find(id);

if (user == null)

{

return HttpNotFound();

}

ViewBag.IDRole = new SelectList(db.Role, "IDRole", "NameRole", user.IDRole);

return View(user);

}

//

// POST: /User/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(User user)

{

if (ModelState.IsValid)

{

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

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.IDRole = new SelectList(db.Role, "IDRole", "NameRole", user.IDRole);

return View(user);

}

//

// GET: /User/Delete/5

public ActionResult Delete(int id = 0)

{

User user = db.User.Find(id);

if (user == null)

{

return HttpNotFound();

}

return View(user);

}

//

// POST: /User/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

User user = db.User.Find(id);

db.User.Remove(user);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

db.Dispose();

base.Dispose(disposing);

}

}

}

3) Разрабатываю авторизацию в системе. Для этого создаю 2 класса (MyRoleProvider, MyMembershipProvider), где описан основной функционал для проверок ролей и паролей.

**MyRoleProvider:**

using kr\_avt.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Security;

public class MyRoleProvider : RoleProvider

{

public static bool Role(string username, string roleName)

{

bool outputResult = false;

// Находим пользователя

using (BDEntities \_db = new BDEntities())

{

// Получаем пользователя

User user = (from u in \_db.User

where u.Login == username

select u).FirstOrDefault();

if (user != null)

{

// получаем роль

var role = \_db.Role.Find(user.IDRole);

//сравниваем

if (role.NameRole.Equals(roleName))

{

outputResult = true;

}

}

}

return outputResult;

}

public override void AddUsersToRoles(string[] usernames, string[] roleNames)

{

throw new NotImplementedException();

}

public override string ApplicationName

{

get

{

throw new NotImplementedException();

}

set

{

throw new NotImplementedException();

}

}

public override void CreateRole(string roleName)

{

throw new NotImplementedException();

}

public override bool DeleteRole(string roleName, bool throwOnPopulatedRole)

{

throw new NotImplementedException();

}

public override string[] FindUsersInRole(string roleName, string usernameToMatch)

{

throw new NotImplementedException();

}

public override string[] GetAllRoles()

{

throw new NotImplementedException();

}

public override string[] GetRolesForUser(string username)

{

throw new NotImplementedException();

}

public override string[] GetUsersInRole(string roleName)

{

throw new NotImplementedException();

}

public override bool IsUserInRole(string username, string roleName)

{

bool outputResult = false;

// Находим пользователя

using (BDEntities \_db = new BDEntities())

{

// Получаем пользователя

User user = (from u in \_db.User

where u.Login == username

select u).FirstOrDefault();

if (user != null)

{

// получаем роль

var role = user.Role;

//сравниваем

if (role != null && role.NameRole == roleName)

{

outputResult = true;

}

}

}

return outputResult;

}

public override void RemoveUsersFromRoles(string[] usernames, string[] roleNames)

{

throw new NotImplementedException();

}

public override bool RoleExists(string roleName)

{

throw new NotImplementedException();

}

}

**MyMembershipProvider:**

using kr\_avt.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Security;

public class MyMembershipProvider : MembershipProvider

{

public override string ApplicationName

{

get

{

throw new NotImplementedException();

}

set

{

throw new NotImplementedException();

}

}

public override bool ChangePassword(string username, string oldPassword, string newPassword)

{

throw new NotImplementedException();

}

public override bool ChangePasswordQuestionAndAnswer(string username, string password, string newPasswordQuestion, string newPasswordAnswer)

{

throw new NotImplementedException();

}

public override MembershipUser CreateUser(string username, string password, string email, string passwordQuestion, string passwordAnswer, bool isApproved, object providerUserKey, out MembershipCreateStatus status)

{

throw new NotImplementedException();

}

public override bool DeleteUser(string username, bool deleteAllRelatedData)

{

throw new NotImplementedException();

}

public override bool EnablePasswordReset

{

get { throw new NotImplementedException(); }

}

public override bool EnablePasswordRetrieval

{

get { throw new NotImplementedException(); }

}

public override MembershipUserCollection FindUsersByEmail(string emailToMatch, int pageIndex, int pageSize, out int totalRecords)

{

throw new NotImplementedException();

}

public override MembershipUserCollection FindUsersByName(string usernameToMatch, int pageIndex, int pageSize, out int totalRecords)

{

throw new NotImplementedException();

}

public override MembershipUserCollection GetAllUsers(int pageIndex, int pageSize, out int totalRecords)

{

throw new NotImplementedException();

}

public override int GetNumberOfUsersOnline()

{

throw new NotImplementedException();

}

public override string GetPassword(string username, string answer)

{

throw new NotImplementedException();

}

public override MembershipUser GetUser(string username, bool userIsOnline)

{

throw new NotImplementedException();

}

public override MembershipUser GetUser(object providerUserKey, bool userIsOnline)

{

throw new NotImplementedException();

}

public override string GetUserNameByEmail(string email)

{

throw new NotImplementedException();

}

public override int MaxInvalidPasswordAttempts

{

get { throw new NotImplementedException(); }

}

public override int MinRequiredNonAlphanumericCharacters

{

get { throw new NotImplementedException(); }

}

public override int MinRequiredPasswordLength

{

get { throw new NotImplementedException(); }

}

public override int PasswordAttemptWindow

{

get { throw new NotImplementedException(); }

}

public override MembershipPasswordFormat PasswordFormat

{

get { throw new NotImplementedException(); }

}

public override string PasswordStrengthRegularExpression

{

get { throw new NotImplementedException(); }

}

public override bool RequiresQuestionAndAnswer

{

get { throw new NotImplementedException(); }

}

public override bool RequiresUniqueEmail

{

get { throw new NotImplementedException(); }

}

public override string ResetPassword(string username, string answer)

{

throw new NotImplementedException();

}

public override bool UnlockUser(string userName)

{

throw new NotImplementedException();

}

public override void UpdateUser(MembershipUser user)

{

throw new NotImplementedException();

}

public override bool ValidateUser(string username, string password)

{

using (BDEntities db = new BDEntities())

{

var result = from u in db.User where (u.Login == username) select u;

if (result.Count() != 0)

{

var dbuser = result.First();

return true;

}

else

{

return false;

}

}

}

}

4) Создаю модель AccountModel, где описаны get и set методы для работы с данными.

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

using System.Data.Entity;

using System.Linq;

using System.Web;

namespace kr\_avt.Models

{

public class UsersContext : DbContext

{

public UsersContext()

: base("BDEntities")

{

}

public DbSet<User> User { get; set; }

public DbSet<Role> Roles { get; set; }

}

[Table("User")]

public class userprofile

{

[Key]

[DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]

public int userid { get; set; }

public string username { get; set; }

}

public class LoginModel

{

[Required]

[Display(Name = "Имя пользователя")]

public string UserName { get; set; }

[Required]

[DataType(DataType.Password)]

[Display(Name = "Пароль")]

public string Password { get; set; }

[Display(Name = "Запомнить меня")]

public bool RememberMe { get; set; }

}

}

5) На основе модели AccountModel создаю контроллер AccountController, где описана логика функционала авторизации

using kr\_avt.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using System.Web.Security;

namespace kr\_avt.Controllers

{

[Authorize]

[AllowAnonymous]

public class AccountController : Controller

{

//

// GET: /Account/

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

[HttpPost]

public ActionResult Login(LoginModel model, string returnUrl)

{

if (ModelState.IsValid)

{

if (Membership.ValidateUser(model.UserName, model.Password))

{

FormsAuthentication.SetAuthCookie(model.UserName, model.RememberMe);

if (Url.IsLocalUrl(returnUrl))

{

return Redirect(returnUrl);

}

else

{

//if (MyRoleProvider.Role(User.Identity.Name, "Администратор") || MyRoleProvider.Role(User.Identity.Name, "Директор"))

//{

// return RedirectToAction("Index", "User");

//}

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

}

}

}

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

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

return View(model);

}

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

FormsAuthentication.SignOut();

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

}

}

}

6) Создаю контроллер HomeController, позволяющий работать с функционалом других контроллеров и представлениями, отображающие основные страницы приложений

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.Entity;

using System.Linq;

using System.Web;

using System.Web.Mvc;

using kr\_avt.Models;

namespace kr\_avt.Controllers

{

public class HomeController : Controller

{

private BDEntities db = new BDEntities();

//

// GET: /Home/

public ActionResult Index()

{

var user = db.User.Include(u => u.Role);

return View(user.ToList());

}

//

// GET: /Home/Details/5

public ActionResult Details(int id = 0)

{

User user = db.User.Find(id);

if (user == null)

{

return HttpNotFound();

}

return View(user);

}

//

// GET: /Home/Create

public ActionResult Create()

{

ViewBag.IDRole = new SelectList(db.Role, "IDRole", "NameRole");

return View();

}

//

// POST: /Home/Create

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Create(User user)

{

if (ModelState.IsValid)

{

db.User.Add(user);

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.IDRole = new SelectList(db.Role, "IDRole", "NameRole", user.IDRole);

return View(user);

}

//

// GET: /Home/Edit/5

[Authorize(Roles="Administrator")]

public ActionResult Edit(int id = 0)

{

User user = db.User.Find(id);

if (user == null)

{

return HttpNotFound();

}

ViewBag.IDRole = new SelectList(db.Role, "IDRole", "NameRole", user.IDRole);

return View(user);

}

//

// POST: /Home/Edit/5

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult Edit(User user)

{

if (ModelState.IsValid)

{

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

db.SaveChanges();

return RedirectToAction("Index");

}

ViewBag.IDRole = new SelectList(db.Role, "IDRole", "NameRole", user.IDRole);

return View(user);

}

//

// GET: /Home/Delete/5

public ActionResult Delete(int id = 0)

{

User user = db.User.Find(id);

if (user == null)

{

return HttpNotFound();

}

return View(user);

}

//

// POST: /Home/Delete/5

[HttpPost, ActionName("Delete")]

[ValidateAntiForgeryToken]

public ActionResult DeleteConfirmed(int id)

{

User user = db.User.Find(id);

db.User.Remove(user);

db.SaveChanges();

return RedirectToAction("Index");

}

protected override void Dispose(bool disposing)

{

db.Dispose();

base.Dispose(disposing);

}

public ActionResult Contact()

{

//ViewBag.Message = "Your contact page.";

return View();

}

public ActionResult HomePage()

{

//ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC application.";

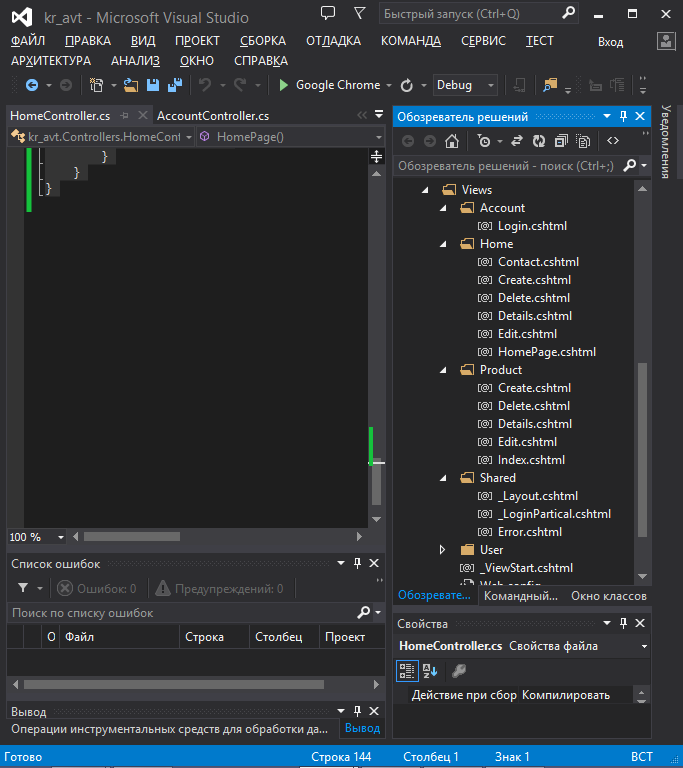
return View();

}

}

}

7) Автоматически создаю представления по функционалу всех контроллеров



8) Запускаю проект (В системе использую роль Администратора)

