*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине***

***«Технологии программирования»***

***на тему***

*Проектирование и разработка программной системы «Кадастровое управление»*

*Выполнил: Яковлева Е.А.*

*Принял: доц. кафедры ИСПИ*

*Вершинин В.В.*

*Владимир, 2022*

**Аннотация**

Целью курсового проектирования является приобретение практических навыков по созданию информационной системы. В данном проекте производилась разработка информационной системы «Кадастровое управление». Для реализации ИС использовалась среда разработки Visual Studio, для работы с базой данных использовался Microsoft SQL. В рамках работы были освоены работа с большим объёмом информации, базой данных, представлениями, контроллерами.

The purpose of the course design is to acquire practical skills in creating an information system. In this project, the information system "Cadastral Management" was developed. The Visual Studio development environment was used to implement the IS, Microsoft SQL was used to work with the database. As part of the work, work with a large amount of information, a database, views, controllers were mastered.

Содержание

[ВВЕДЕНИЕ 4](#_Toc122024491)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc122024492)

[1.1 Словарь предметной области 5](#_Toc122024493)

[1.2 Взаимоотношения между классами 6](#_Toc122024494)

[1.3 Диаграмма прецедентов 6](#_Toc122024495)

[1.4 Расширенное описание прецедентов 7](#_Toc122024496)

[1.5 Диаграмма классов 8](#_Toc122024497)

[1.6 Диаграмма состояний 8](#_Toc122024498)

[1.7 Диаграмма последовательностей 9](#_Toc122024499)

[2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ 10](#_Toc122024500)

[2.1 Общие принципы организации системы 10](#_Toc122024501)

[2.2 Реализация модели 10](#_Toc122024502)

[2.3 Создание заявления на земельный кадастр 11](#_Toc122024503)

[2.4 Одобрение заявления на земельный кадастр 11](#_Toc122024504)

[2.5 Отказ в регистрации земельного кадастра. 12](#_Toc122024505)

[2.6 Вывод заявлений авторизированных пользователей 13](#_Toc122024506)

[2.7 Оплата налогов 14](#_Toc122024507)

[ЗАКЛЮЧЕНИЕ 16](#_Toc122024508)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 17](#_Toc122024509)

[ПРИЛОЖЕНИЕ А 18](#_Toc122024510)

[ИСХОДНЫЙ КОД ПРОГРАММНОЙ СИСТЕМЫ 18](#_Toc122024511)

[ПРИЛОЖЕНИЕ Б 56](#_Toc122024512)

[СТРУКТУРА ТАБЛИЦЫ БД 56](#_Toc122024513)

## ВВЕДЕНИЕ

Система кадастрового управления призвана организовывать процесс учета кадастровых объектов с использованием информационных технологий. Кадастр – это свод данных, в котором фиксируется вся информация об объектах недвижимости. Современные технологии позволяют создать систему для кадастрового управления. Программно-информационная система позволит создавать кадастровые объекты с понятным интерфейсом.

Цель данного курсового проекта является разработка программной информационной системы «Кадастровое управление».

Функциональные требования:

- ввод данных в систему о кадастровых объектах;

- работа со справочником типов объектов учета в Кадастровом управлении;

- самостоятельная регистрация заявителя в системе;

- подача заявки на регистрацию кадастрового объекта;

- рассмотрение заявок и регистрация (или отказ);

- просмотр сведений о кадастровых объектах и изменение сведений о них;

- оплата налога за кадастровый объект авторизованным пользователем.

## 1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ

В данной работе рассматривается система управления кадастрового учета. Ее назначение: хранение, дополнение и изменение кадастра. В систему входят данные о пользователях, различных типах кадастровых объектов (земельных, недвижимостях, лесах, водных, полезных ископаемых) и их параметрах.

Пользователями данной системы могут быть собственники или потенциальные собственники имущества. Управлять данной системой будут федеральные службы государственной регистрации кадастра и картографии.

Возможности пользователя: подача заявки на регистрацию кадастрового объекта, просмотр сведений о кадастровых объектах, оплата налогов на кадастровые объекты.

Возможности кадастрового агента: просмотр сведений о кадастровых объектах, изменение сведений о кадастровых объектах, рассмотрение заявок и регистрация или отказ регистрации кадастрового объекта, ввод данных в систему, работа со справочником типов объектов учета в Кадастровом управлении, а также изменение и добавление налогов на кадастровые объекты.

1.1 Словарь предметной области

*Гость (Guest)* – пользователь, не зарегистрированный в системе.

*Авторизованный* *пользователь (AutorizedUser)* – пользователь, который вошел в систему, предварительно зарегистрировавшись в ней.

*Атрибуты*: имя (Name), паспортные данные (PassportData), пароль (Password).

*Методы*: добавить пользователя (Adding), удалить пользователя (Delete).

*Справочник* *типов (Directory)* – описание видов кадастровых объектов.

*Атрибуты*: тип объекта (Type), описание (Notes).

*Методы*: поиск по типу (Search), изменение (Change).

*Кадастровый* *агент (CadastralAgent)* – авторизованный пользователь, имеющий дополнительные возможности в системе.

*Атрибуты*: имя (Name), логин (Login), пароль (Password).

*Методы*: добавление кадастрового агента (Adding), удаление кадастрового агента (Delete).

*Кадастр (Cadastre)* – документ, включающий в себя список всех зарегистрированных кадастровых объектов.

*Методы*: просмотр кадастра (viewCadastre).

*Земельный* *кадастр (LandRegistry)* – документ, содержащий список всех кадастровых объектов относящихся к земельному типу объектов.

*Атрибуты*: площадь земельного участка (Area), адрес (Address), кадастровая стоимость (Price), дата регистрации (ApprovalDate), паспортные данные владельца (Passport).

*Методы*: поиск по паспортным данным владельца (Search), поиск по ID(Search), изменение владельца объекта (Change), изменение характеристик объекта (Change), добавление объекта (Adding), удаление объекта (Delete).

*Кадастр* *недвижимости (RealEstateCadastre)* - документ, содержащий список всех кадастровых объектов относящихся к типу объектов - недвижимость.

*Атрибуты*: тип недвижимости (Type), площадь (Area), адрес (Address), дата регистрации (ApprovalDate), паспортные данные владельца (Passport), степень готовности (Readiness), назначение (Appointment), этажность (Floors), дополнительная информация (Notes).

*Методы*: поиск по паспортным данным владельца (Search), поиск по ID (Search), изменение владельца объекта (Change), изменение характеристик объекта (Change), добавление объекта (Adding), удаление объекта (Delete).

*Водный* *кадастр (WaterCadastre)* - документ, содержащий список всех кадастровых объектов относящихся к типу водных объектов.

*Атрибуты*: площадь водного участка (Area), адрес (Address), тип водного объекта (Type), дата регистрации (ApprovalDate), паспортные данные владельца (Passport).

*Методы*: поиск по паспортным данным владельца (Search), поиск по ID (Search), изменение владельца объекта (Change), изменение характеристик объекта (Change), добавление объекта (Adding), удаление объекта (Delete).

*Кадастр* *природных* *ископаемых (Minerals)* - документ, содержащий список всех кадастровых объектов относящихся к типу природных ископаемых.

*Атрибуты*: адрес (Address), тип природных ископаемых (Type), дата регистрации (ApprovalDate), паспортные данные владельца (Passport), объем (Amount), кадастровая стоимость (Price).

*Методы*: поиск по паспортным данным владельца (Change), поиск по ID (Search), изменение владельца объекта (Change), изменение характеристик объекта (Change), добавление объекта (Adding), удаление объекта (Delete).

*Заявление (Statement)* – прошение на регистрацию кадастрового объекта в кадастр.

*Атрибуты*: статус заявления (status), тип кадастрового объекта (type).

*Методы*: поиск по ID (Search), добавление заявления (Adding), удаление заявления (Delete).

*Налог (Tax)* - обязательный, индивидуально безвозмездный платёж, взимаемый с организаций и физических лиц в форме отчуждения принадлежащих им на праве собственности средств, в целях финансового обеспечения деятельности государства и муниципальных образований.

*Атрибуты*: цена (Price), статус налога (Tax), дата создания (DataOfCreation), срок (Term).

1.2 Взаимоотношения между классами

*Авторизованный пользователь* и *кадастровый агент* наследники *гостя*.

*Гость* может просматривать *справочник* *типов* (ассоциация).

*Гость* может просматривать *кадастр* (ассоциация).

*Авторизованный пользователь* может создавать и просматривать *заявления* (ассоциация).

*Кадастр* состоит в отношениях агрегации с *земельным* *кадастром, кадастром недвижимости, водным кадастром и кадастром природных ископаемых*.

*Кадастровый агент* может изменять *справочник типов* (ассоциация).

*Кадастровый агент* может изменять *кадастр* (ассоциация).

*Кадастровый агент* может рассматривать *заявления* (ассоциация).

1.3 Диаграмма прецедентов

Диаграмма прецедентов представлена на рисунке 1.



Рисунок 1. Диаграмма прецедентов.

1.4 Расширенное описание прецедентов

Расширенное описание прецедента «Рассмотреть заявку на регистрацию кадастрового объекта»:

**Название**: «Рассмотреть заявку на регистрацию кадастрового объекта»

**Предусловие**: кадастровый агент зарегистрирован в ИС «Кадастровое управление», т.е. авторизован в системе.

**Действующее** **лицо**: кадастровый агент

**Основной** **поток**: *Рассмотреть заявку на регистрацию кадастрового объекта*

Кадастровый агент открывает форму, отображающую личные данные заявителя, данные кадастрового объекта. После принятия решения о регистрации объекта кадастровый агент либо переход к регистрации объекта с помощью кнопки «Принять заявку», либо отказ от регистрации с помощью кнопки «Отклонить заявку».

При принятии решения о регистрации объекта данные из заявки сохраняются в БД и перенаправляются в следующую форму (вариант использования «Внести данные в систему»).

**Альтернативный** **поток**: отказ от регистрации.

Если пришлось отказать в регистрации объекта, то об этом решении оповещают заявителя.

**Постусловие**: если принято решение о принятие заявки, то кадастровый агент переходит к регистрации объекта.

Расширенное описание прецедента «Изменить сведения о кадастровом объекте»:

**Название**: «Изменить сведения о кадастровом объекте»

**Предусловие**: кадастровый агент зарегистрирован в ИС «Кадастровое управление», т.е. авторизован в системе.

**Действующее** **лицо**: кадастровый агент

**Основной** **поток**: *Изменить сведения о кадастровом объекте*

Кадастровый агент открывает форму, отображающую список кадастровых объектов. Пользуется средствами поиска (вводит имя владельца или параметры объекта).

Изменяет необходимые данные о владельце или параметрах кадастрового объекта.

Система сохраняет изменения описания кадастрового объекта.

**Альтернативный** **поток**: запись о кадастровом объекте отсутствует.

Если запись о кадастровом объекте отсутствует, то кадастровый агент либо создает новую запись о кадастровом объекте, либо корректирует условия поиска.

**Постусловие**: если запись найдена, то вносятся изменения в БД.

Расширенное описание прецедента «Просмотреть статус заявления»:

**Название**: «Просмотреть статус заявления»

**Предусловие**: авторизованный пользователь зарегистрирован в ИС «Кадастровое управление», т.е. авторизован в системе.

**Действующее** **лицо**: авторизованный пользователь

**Основной** **поток**: *Просмотреть статус заявления*

Авторизованный пользователь открывает форму, отображающую список заявлений. После каждого заявления стоит отметка статуса заявления в данный момент (обрабатывается, одобрено, отказано).

1.5 Диаграмма классов

Диаграмма классов представлена на рисунке 2.



Рисунок 2. Диаграмма классов.

1.6 Диаграмма состояний

Рассмотрим использование диаграмм состояний на примере состояний объекта «Заявление» в ИС «Кадастровое управление». Один из возможных процессов работы ИС с заявлением представлен на рисунке 2.



Рисунок 3. Диаграмма состояний для заявления.

Описание состояний, переходов:

1. Пользователь оформил заявление, указав свои данные и данные кадастрового объекта. Объект «Заявление» создано в системе (объект в записи БД).
2. От кадастрового агента пришло сообщение о подтверждении или опровержении данных.
3. От кадастрового агента пришло сообщение о подтверждении данных. Объект «Заявление» отмечается как принятое.
4. От кадастрового агента пришло сообщение об опровержении данных. Заявление помечается отказанным. После этого переходим к п. 8.
5. От кадастрового агента пришло сообщение о принятие решении о разрешении либо запрете регистрации кадастрового объекта.
6. От кадастрового агента пришло сообщение о принятие решения о запрете регистрации кадастрового объекта. Объект «Заявление» помечается как отказанный. После этого переходим к п. 8.
7. От кадастрового агента пришло сообщение о принятие решения о разрешении регистрации кадастрового объекта. Объект «Заявление» помечается как зарегистрированное.
8. Заявление помечается как отклоненное.

1.7 Диаграмма последовательностей

Описание взаимодействий:

1. Пользователь авторизуется в системе для доступа к оформлению заявлений.
2. Информационная система открывает приветственную форму.
3. Пользователь при помощи кнопки отправляет запрос на оформление заявления.
4. Информационная система открывает форму для формирования заявления.
5. Пользователь вводит личные данные в нужные строки формы заявления.
6. Пользователь вводит данные кадастрового объекта в нужные строки формы заявления.
7. Форма заявления проходит валидацию внесенных данных.
8. Форма заявления оформляет заявление.
9. Заявление заносится в базу данных.
10. Информационная система отправляет сообщение клиенту о приеме заявления.

Диаграмма последовательностей для представлена на рисунке 3.



Рисунок 4. Диаграмма последовательностей.

## 2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1 Общие принципы организации системы

Разрабатываемая программная система представляет собой веб-приложение, которое построено на базе архитектуры ASP.NET Fraemwork MVC.

Приложение MVC реализует шаблон, состоящий из трех компонентов: контроллера, модели и представления или Model-View-Controller. Каждый контроллер относится к своему классу модели и содержит необходимые методы, которые реализуют бизнес-функции и в результате своей работы возвращают пользователю представление, которое отображает результат выполнения операций. В контроллерах используется внедрение зависимостей – это механизм отделения конструирования от использования, что способствует соблюдению принципа единой ответственности. Для работы с данными используется экземпляр класса DataClasses1, наследника класса System.Data.Linq.DataContext, представляющий основную точку входа для платформы LINQ to SQL.

В ходе проектирования системы, было принято решение, использовать СУБД MS SQL Server, ввиду её хорошей совместимости со средой разработки Visual Studio. Является лучшим выбором для широкого спектра конечных пользователей и программистов баз данных, создающих бизнес-приложения.

2.2 Реализация модели

Для реализации данной системы был выбран подход Database-First. При этом подходе, рабочий процесс создания модели начинается с создания и проектирования базы данных. Была создана база данных CadastreDB. Все данные, при необходимости, подгружаются с помощью фреймворка из базы данных. Физическая модель базы данных представлена в приложении.

2.3 Создание заявления на земельный кадастр

При создании заявления автоматически ставим его статус «Обрабатывается». Создать заявление может как кадастровый агент, так и авторизированный пользователь. Создание представлено на рисунке 4.

[HttpPost]

[Authorize(Roles = "user,admin")]

public ActionResult Create(StatementLand collection)

{

try

{

collection.Status = "Обрабатывается";

dc.StatementLand.InsertOnSubmit(collection);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

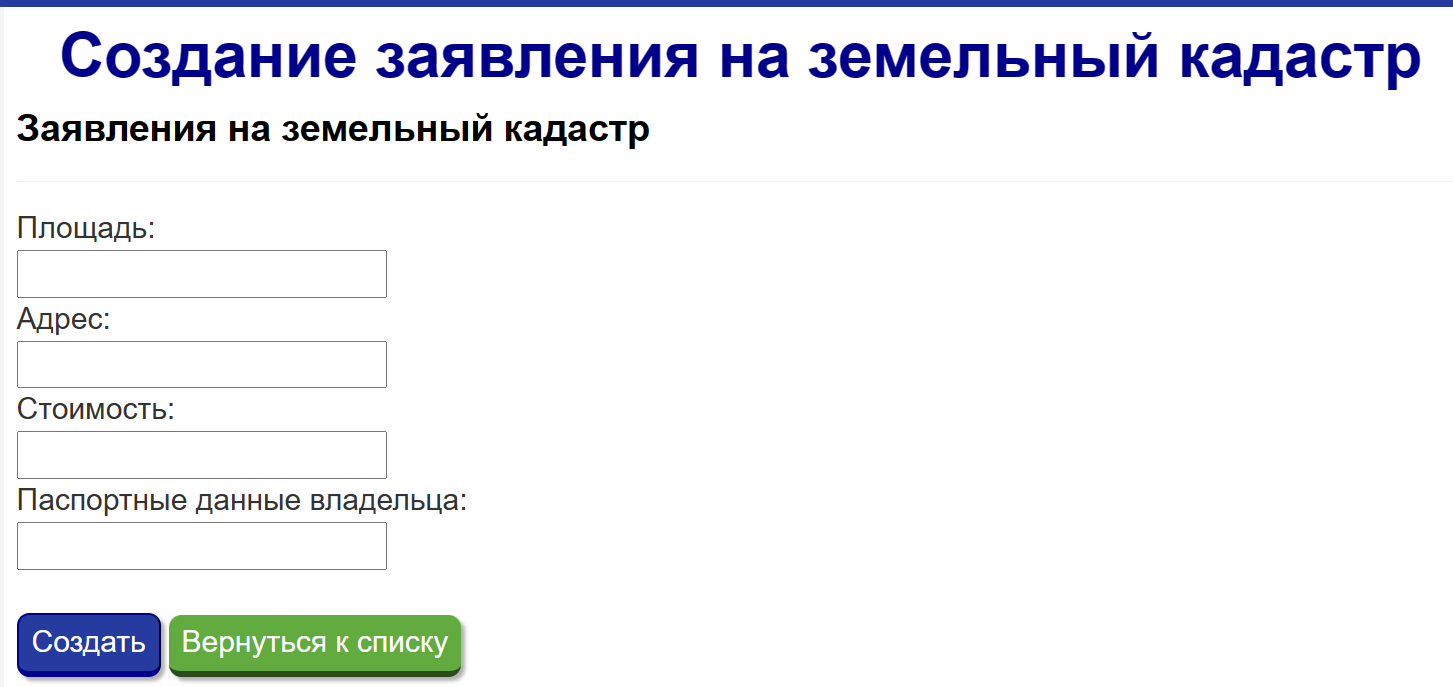


Рисунок 5. Создание заявления на земельный кадастр.

2.4 Одобрение заявления на земельный кадастр

При одобрении заявления на земельный кадастр агентом создается новая запись о земельном кадастре с данными из заявления, и изменяется статус заявления на «Одобрено».

public ActionResult CreateLand(int id, StatementLand collection)

{

LandRegistry statementLand = new LandRegistry();

StatementLand statement = dc.StatementLand.Single(x => x.Id == id);

statementLand.Area = statement.Area;

statementLand.Address = statement.Address;

statementLand.Price = statement.Price;

statementLand.ApprovalDate = DateTime.Today;

statementLand.Passport = statement.Passport;

const double V = 0.2;

statementLand.Tax\_coefficint =(float) V;

dc.LandRegistry.InsertOnSubmit(statementLand);

statement.Status = "Одобрено";

dc.SubmitChanges();

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

}

****

Рисунок 6. Таблица заявлений у авторизированного пользователя до одобрения.

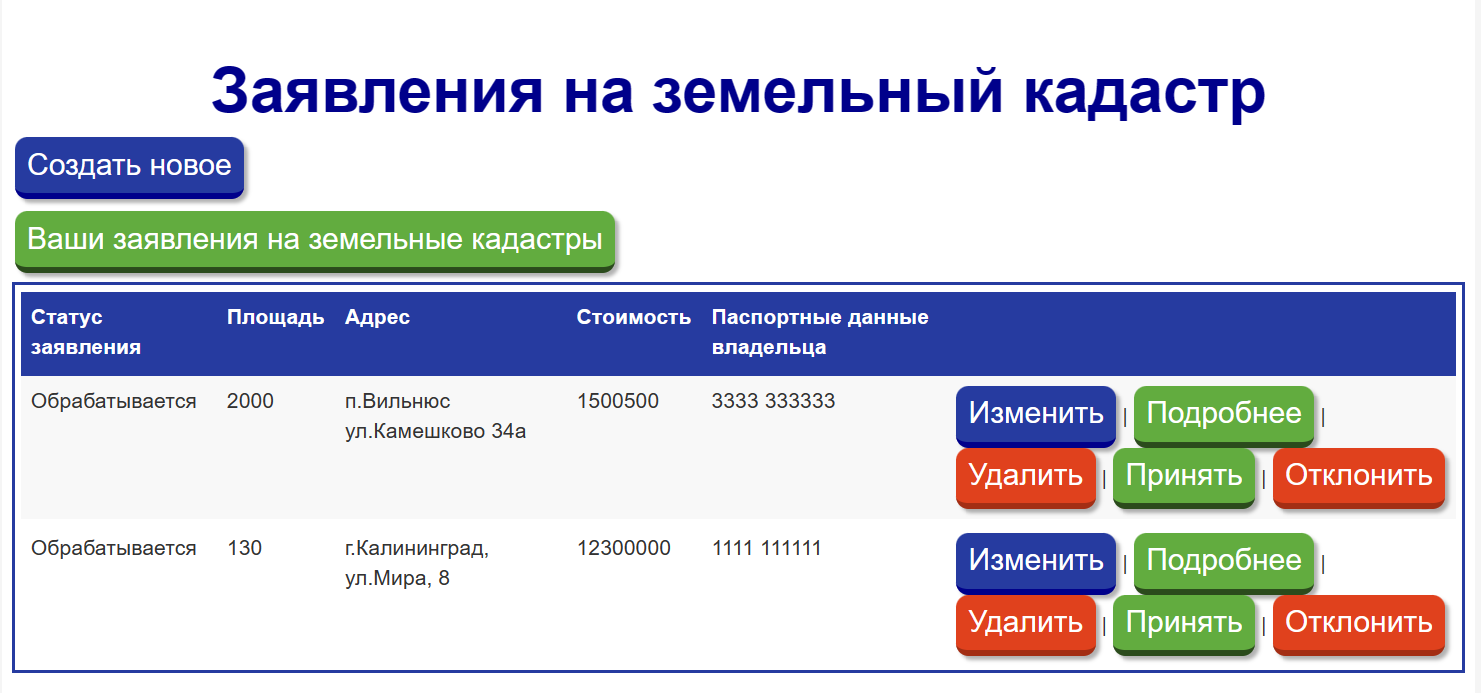


Рисунок 7. Таблица заявлений у кадастрового агента.



Рисунок 8. Таблица заявлений после одобрения.

2.5 Отказ в регистрации земельного кадастра.

При отказе регистрации земельного кадастра агентом изменяется статус заявления на «Отказано».

public ActionResult RefusalLand(int id, StatementLand collection)

{

// TODO: Add insert logic here

StatementLand statement = dc.StatementLand.Single(x => x.Id == id);

statement.Status = "Отказано";

dc.SubmitChanges();

return RedirectToAction("Index");

}



Рисунок 9. Таблица заявлений у авторизированного пользователя до одобрения.

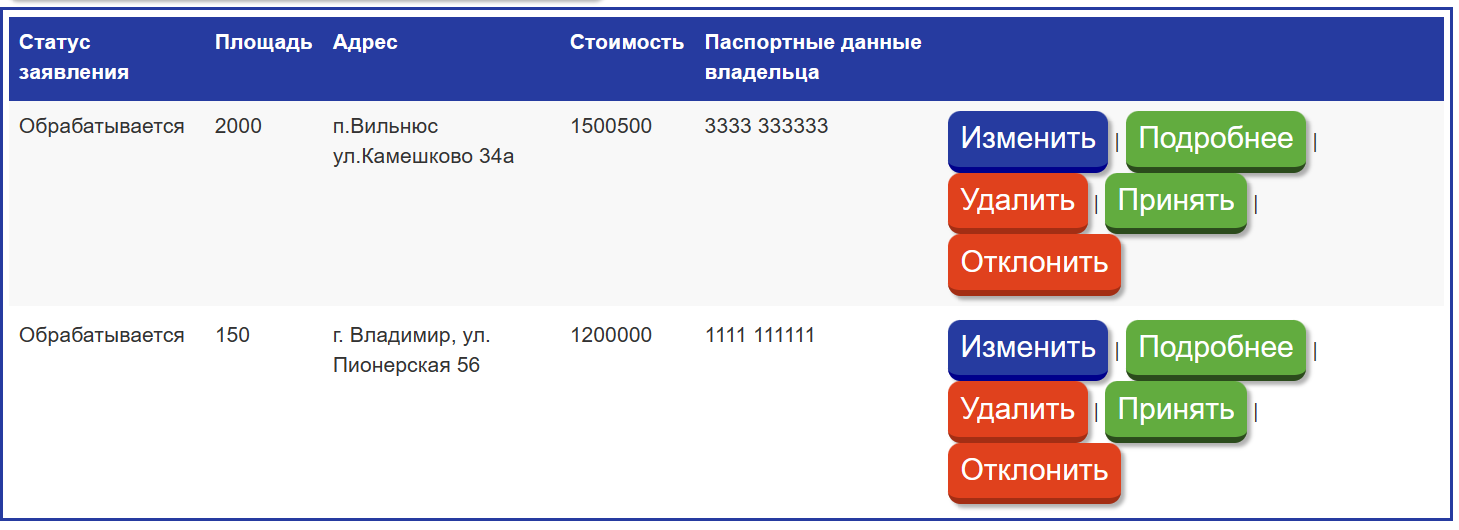


Рисунок 10. Таблица заявлений у кадастрового агента.

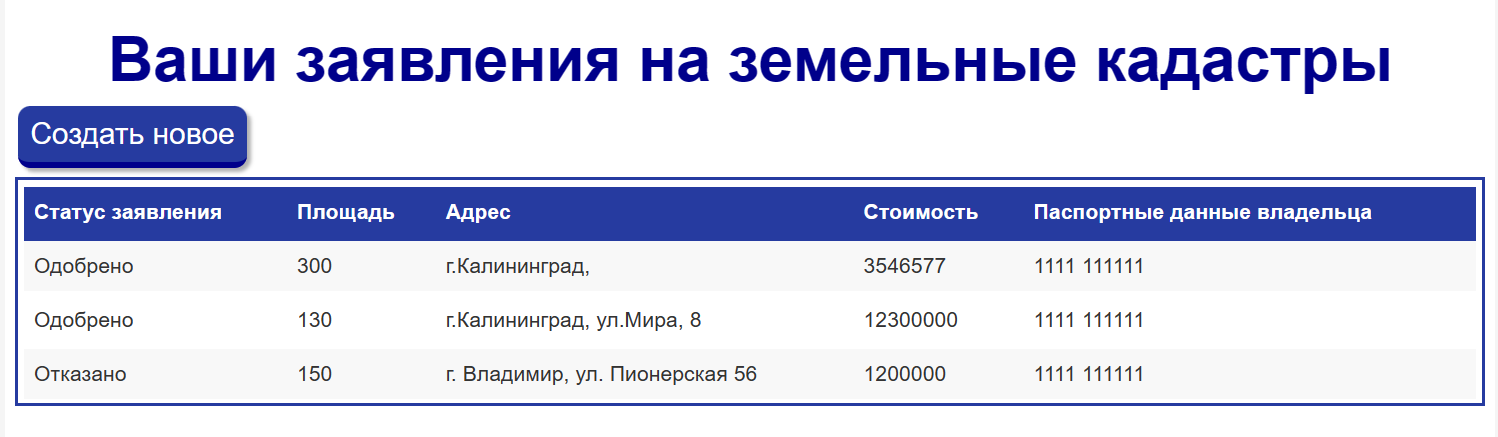


Рисунок 11. Таблица заявлений у авторизированного пользователя до одобрения.

2.6 Вывод заявлений авторизированных пользователей

Вывод заявлений авторизированных пользователей осуществляется при помощи выборки записей по id владельца.

public ActionResult IndexId(string userId)

{

List<StatementLand> statementLandDetails = new List<StatementLand>();

statementLandDetails = dc.StatementLand.Where(x => x.Passport == userId).ToList();

return View(statementLandDetails);

}

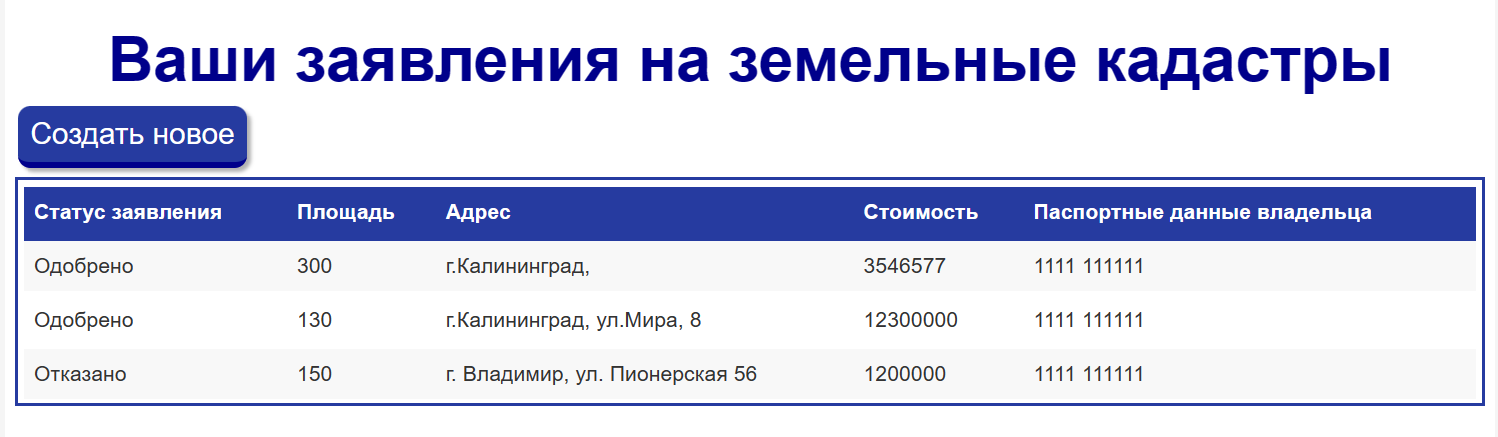


Рисунок 12. Вывод заявлений авторизированного пользователя.

2.7 Оплата налогов

При оплате налога на кадастровый объект изменяется дата оплат налога на сегодняшнюю.

[HttpPost]

public ActionResult EditPaymentDate(int id, Tax collection)

{

try

{

// TODO: Add update logic here

Tax tax = dc.Tax.Single(x => x.Id == id);

tax.Price = collection.Price;

tax.Date\_of\_creation = collection.Date\_of\_creation;

tax.Payment\_date = DateTime.Today;

tax.Cadastre\_type = collection.Cadastre\_type;

tax.Cadastre\_id = collection.Cadastre\_id;

dc.SubmitChanges();

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

}

catch

{

return View();

}

}

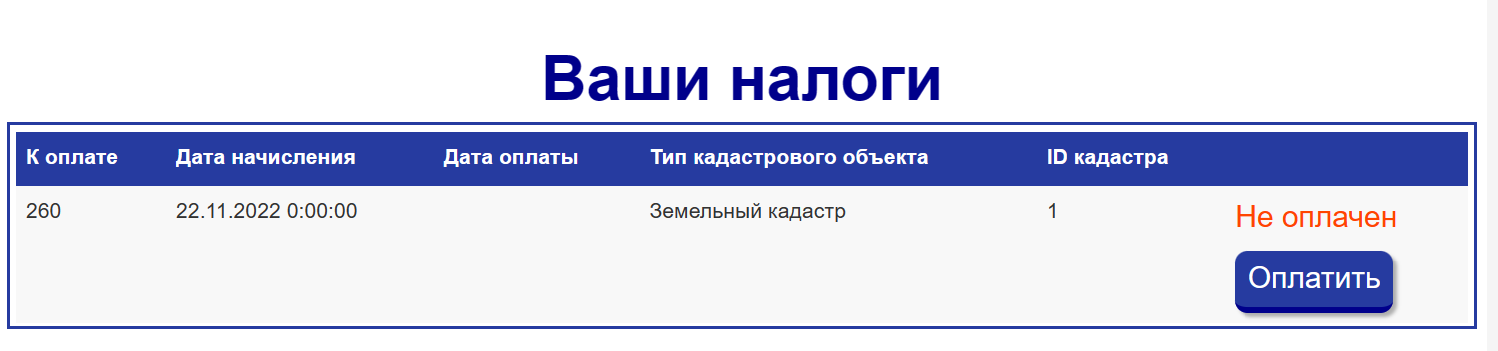


Рисунок 13. Налоги на кадастр до оплаты.

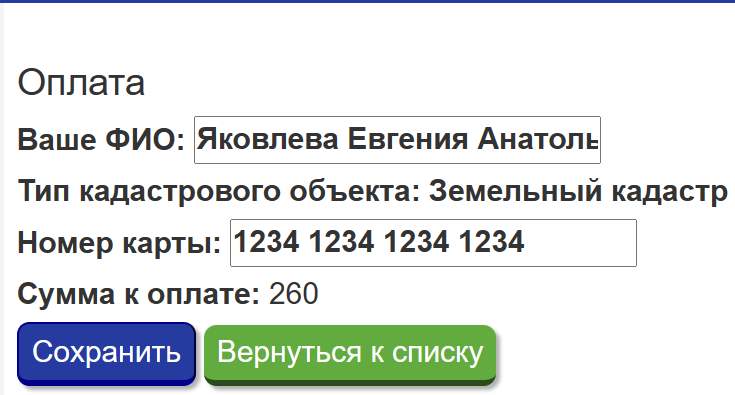


Рисунок 14. Форма для оплаты налога.

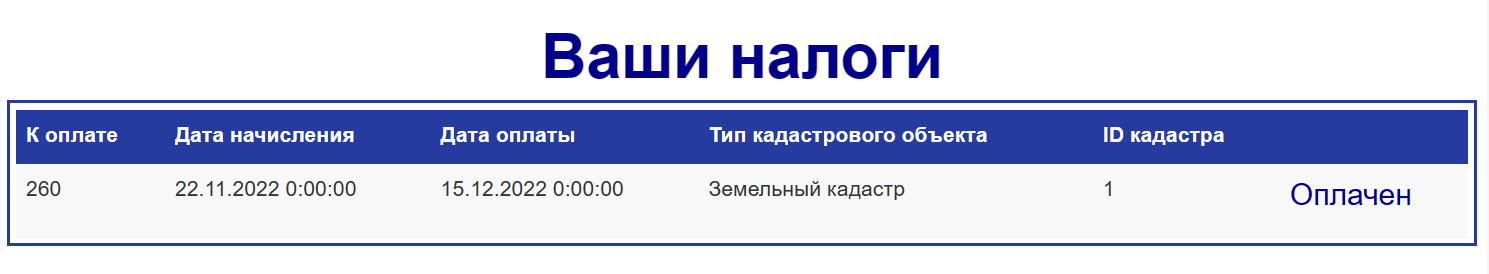


Рисунок 15. Налоги на кадастр после оплаты.

## ЗАКЛЮЧЕНИЕ

В выполнения курсового проекта была разработана программная система «Кадастровое управление». Для удобства использования системы был создан простой интерфейс и соблюдены требования платформы ASP.NET Fraemwork MVC.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

* 1. Липаев В.В. Проектирование программных систем. М.: Высш. шк, 1990.
  2. Буч Г. Объектно-ориентированное проектирование / Пер. с анг. Конкорд, 1996.
  3. Майерс Г. Надежность программного обеспечения. М.: Мир, 1980.\
  4. Руководство по Entity Framework Core 6 [Электронный ресурс] - Режим доступа: <https://metanit.com/sharp/efcore/> - 08.12.2021
  5. Душан Петкович. Microsoft SQL Server 2008. Руководство для начинающих, БХВ-Петербург, 2009 – 752 с.

## ПРИЛОЖЕНИЕ А

## ИСХОДНЫЙ КОД ПРОГРАММНОЙ СИСТЕМЫ

**Контроллеры:**

**AccountController.cs:**

namespace CadastreManagment.Controllers

{

[Authorize]

public class AccountController : Controller

{

private ApplicationSignInManager \_signInManager;

private ApplicationUserManager \_userManager;

public AccountController()

{

}

public AccountController(ApplicationUserManager userManager, ApplicationSignInManager signInManager )

{

UserManager = userManager;

SignInManager = signInManager;

}

public ApplicationSignInManager SignInManager

{

get

{

return \_signInManager ?? HttpContext.GetOwinContext().Get<ApplicationSignInManager>();

}

private set

{

\_signInManager = value;

}

}

public ApplicationUserManager UserManager

{

get

{

return \_userManager ?? HttpContext.GetOwinContext().GetUserManager<ApplicationUserManager>();

}

private set

{

\_userManager = value;

}

}

//

// GET: /Account/Login

[AllowAnonymous]

public ActionResult Login(string returnUrl)

{

ViewBag.ReturnUrl = returnUrl;

return View();

}

//

// POST: /Account/Login

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)

{

if (!ModelState.IsValid)

{

return View(model);

}

// Сбои при входе не приводят к блокированию учетной записи

// Чтобы ошибки при вводе пароля инициировали блокирование учетной записи, замените на shouldLockout: true

var result = await SignInManager.PasswordSignInAsync(model.Email, model.Password, model.RememberMe, shouldLockout: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = model.RememberMe });

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Неудачная попытка входа.");

return View(model);

}

}

//

// GET: /Account/VerifyCode

[AllowAnonymous]

public async Task<ActionResult> VerifyCode(string provider, string returnUrl, bool rememberMe)

{

// Требовать предварительный вход пользователя с помощью имени пользователя и пароля или внешнего имени входа

if (!await SignInManager.HasBeenVerifiedAsync())

{

return View("Error");

}

return View(new VerifyCodeViewModel { Provider = provider, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/VerifyCode

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> VerifyCode(VerifyCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

// Приведенный ниже код защищает от атак методом подбора, направленных на двухфакторные коды.

// Если пользователь введет неправильные коды за указанное время, его учетная запись

// будет заблокирована на заданный период.

// Параметры блокирования учетных записей можно настроить в IdentityConfig

var result = await SignInManager.TwoFactorSignInAsync(model.Provider, model.Code, isPersistent: model.RememberMe, rememberBrowser: model.RememberBrowser);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(model.ReturnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.Failure:

default:

ModelState.AddModelError("", "Неправильный код.");

return View(model);

}

}

//

// GET: /Account/Register

[AllowAnonymous]

public ActionResult Register()

{

return View();

}

//

// POST: /Account/Register

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> Register(RegisterViewModel model)

{

if (ModelState.IsValid)

{

var user = new ApplicationUser { Id = model.Id ,UserName = model.Email, Email = model.Email};

var result = await UserManager.CreateAsync(user, model.Password);

if (result.Succeeded)

{

await UserManager.AddToRoleAsync(user.Id, "user");

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

// Дополнительные сведения о включении подтверждения учетной записи и сброса пароля см. на странице https://go.microsoft.com/fwlink/?LinkID=320771.

// Отправка сообщения электронной почты с этой ссылкой

// string code = await UserManager.GenerateEmailConfirmationTokenAsync(user.Id);

// var callbackUrl = Url.Action("ConfirmEmail", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Подтверждение учетной записи", "Подтвердите вашу учетную запись, щелкнув <a href=\"" + callbackUrl + "\">здесь</a>");

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

}

AddErrors(result);

}

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

return View(model);

}

//

// GET: /Account/ConfirmEmail

[AllowAnonymous]

public async Task<ActionResult> ConfirmEmail(string userId, string code)

{

if (userId == null || code == null)

{

return View("Error");

}

var result = await UserManager.ConfirmEmailAsync(userId, code);

return View(result.Succeeded ? "ConfirmEmail" : "Error");

}

//

// GET: /Account/ForgotPassword

[AllowAnonymous]

public ActionResult ForgotPassword()

{

return View();

}

//

// POST: /Account/ForgotPassword

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ForgotPassword(ForgotPasswordViewModel model)

{

if (ModelState.IsValid)

{

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null || !(await UserManager.IsEmailConfirmedAsync(user.Id)))

{

// Не показывать, что пользователь не существует или не подтвержден

return View("ForgotPasswordConfirmation");

}

// Дополнительные сведения о включении подтверждения учетной записи и сброса пароля см. на странице https://go.microsoft.com/fwlink/?LinkID=320771.

// Отправка сообщения электронной почты с этой ссылкой

// string code = await UserManager.GeneratePasswordResetTokenAsync(user.Id);

// var callbackUrl = Url.Action("ResetPassword", "Account", new { userId = user.Id, code = code }, protocol: Request.Url.Scheme);

// await UserManager.SendEmailAsync(user.Id, "Сброс пароля", "Сбросьте ваш пароль, щелкнув <a href=\"" + callbackUrl + "\">здесь</a>");

// return RedirectToAction("ForgotPasswordConfirmation", "Account");

}

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

return View(model);

}

//

// GET: /Account/ForgotPasswordConfirmation

[AllowAnonymous]

public ActionResult ForgotPasswordConfirmation()

{

return View();

}

//

// GET: /Account/ResetPassword

[AllowAnonymous]

public ActionResult ResetPassword(string code)

{

return code == null ? View("Error") : View();

}

//

// POST: /Account/ResetPassword

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ResetPassword(ResetPasswordViewModel model)

{

if (!ModelState.IsValid)

{

return View(model);

}

var user = await UserManager.FindByNameAsync(model.Email);

if (user == null)

{

// Не показывать, что пользователь не существует

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

var result = await UserManager.ResetPasswordAsync(user.Id, model.Code, model.Password);

if (result.Succeeded)

{

return RedirectToAction("ResetPasswordConfirmation", "Account");

}

AddErrors(result);

return View();

}

//

// GET: /Account/ResetPasswordConfirmation

[AllowAnonymous]

public ActionResult ResetPasswordConfirmation()

{

return View();

}

//

// POST: /Account/ExternalLogin

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public ActionResult ExternalLogin(string provider, string returnUrl)

{

// Запрос перенаправления к внешнему поставщику входа

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

}

//

// GET: /Account/SendCode

[AllowAnonymous]

public async Task<ActionResult> SendCode(string returnUrl, bool rememberMe)

{

var userId = await SignInManager.GetVerifiedUserIdAsync();

if (userId == null)

{

return View("Error");

}

var userFactors = await UserManager.GetValidTwoFactorProvidersAsync(userId);

var factorOptions = userFactors.Select(purpose => new SelectListItem { Text = purpose, Value = purpose }).ToList();

return View(new SendCodeViewModel { Providers = factorOptions, ReturnUrl = returnUrl, RememberMe = rememberMe });

}

//

// POST: /Account/SendCode

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> SendCode(SendCodeViewModel model)

{

if (!ModelState.IsValid)

{

return View();

}

// Создание и отправка маркера

if (!await SignInManager.SendTwoFactorCodeAsync(model.SelectedProvider))

{

return View("Error");

}

return RedirectToAction("VerifyCode", new { Provider = model.SelectedProvider, ReturnUrl = model.ReturnUrl, RememberMe = model.RememberMe });

}

//

// GET: /Account/ExternalLoginCallback

[AllowAnonymous]

public async Task<ActionResult> ExternalLoginCallback(string returnUrl)

{

var loginInfo = await AuthenticationManager.GetExternalLoginInfoAsync();

if (loginInfo == null)

{

return RedirectToAction("Login");

}

// Выполнение входа пользователя посредством данного внешнего поставщика входа, если у пользователя уже есть имя входа

var result = await SignInManager.ExternalSignInAsync(loginInfo, isPersistent: false);

switch (result)

{

case SignInStatus.Success:

return RedirectToLocal(returnUrl);

case SignInStatus.LockedOut:

return View("Lockout");

case SignInStatus.RequiresVerification:

return RedirectToAction("SendCode", new { ReturnUrl = returnUrl, RememberMe = false });

case SignInStatus.Failure:

default:

// Если у пользователя нет учетной записи, то ему предлагается создать ее

ViewBag.ReturnUrl = returnUrl;

ViewBag.LoginProvider = loginInfo.Login.LoginProvider;

return View("ExternalLoginConfirmation", new ExternalLoginConfirmationViewModel { Email = loginInfo.Email });

}

}

//

// POST: /Account/ExternalLoginConfirmation

[HttpPost]

[AllowAnonymous]

[ValidateAntiForgeryToken]

public async Task<ActionResult> ExternalLoginConfirmation(ExternalLoginConfirmationViewModel model, string returnUrl)

{

if (User.Identity.IsAuthenticated)

{

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

}

if (ModelState.IsValid)

{

// Получение сведений о пользователе от внешнего поставщика входа

var info = await AuthenticationManager.GetExternalLoginInfoAsync();

if (info == null)

{

return View("ExternalLoginFailure");

}

var user = new ApplicationUser { UserName = model.Email, Email = model.Email };

var result = await UserManager.CreateAsync(user);

if (result.Succeeded)

{

result = await UserManager.AddLoginAsync(user.Id, info.Login);

if (result.Succeeded)

{

await SignInManager.SignInAsync(user, isPersistent: false, rememberBrowser: false);

return RedirectToLocal(returnUrl);

}

}

AddErrors(result);

}

ViewBag.ReturnUrl = returnUrl;

return View(model);

}

//

// POST: /Account/LogOff

[HttpPost]

[ValidateAntiForgeryToken]

public ActionResult LogOff()

{

AuthenticationManager.SignOut(DefaultAuthenticationTypes.ApplicationCookie);

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

}

//

// GET: /Account/ExternalLoginFailure

[AllowAnonymous]

public ActionResult ExternalLoginFailure()

{

return View();

}

protected override void Dispose(bool disposing)

{

if (disposing)

{

if (\_userManager != null)

{

\_userManager.Dispose();

\_userManager = null;

}

if (\_signInManager != null)

{

\_signInManager.Dispose();

\_signInManager = null;

}

}

base.Dispose(disposing);

}

#region Вспомогательные приложения

// Используется для защиты от XSRF-атак при добавлении внешних имен входа

private const string XsrfKey = "XsrfId";

private IAuthenticationManager AuthenticationManager

{

get

{

return HttpContext.GetOwinContext().Authentication;

}

}

private void AddErrors(IdentityResult result)

{

foreach (var error in result.Errors)

{

ModelState.AddModelError("", error);

}

}

private ActionResult RedirectToLocal(string returnUrl)

{

if (Url.IsLocalUrl(returnUrl))

{

return Redirect(returnUrl);

}

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

}

internal class ChallengeResult : HttpUnauthorizedResult

{

public ChallengeResult(string provider, string redirectUri)

: this(provider, redirectUri, null)

{

}

public ChallengeResult(string provider, string redirectUri, string userId)

{

LoginProvider = provider;

RedirectUri = redirectUri;

UserId = userId;

}

public string LoginProvider { get; set; }

public string RedirectUri { get; set; }

public string UserId { get; set; }

public override void ExecuteResult(ControllerContext context)

{

var properties = new AuthenticationProperties { RedirectUri = RedirectUri };

if (UserId != null)

{

properties.Dictionary[XsrfKey] = UserId;

}

context.HttpContext.GetOwinContext().Authentication.Challenge(properties, LoginProvider);

}

}

#endregion

}

}

**DirectoryController.cs:**

namespace CadastreManagment.Controllers

{

public class DirectoryController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: Directory

public ActionResult Index()

{

var directoriDetails = from x in dc.Directory select x;

return View(directoriDetails);

}

// GET: Directory/Details/5

public ActionResult Details(string type)

{

var getDirectoryDetails = dc.Directory.Single(x => x.Type == type);

return View(getDirectoryDetails);

}

// GET: Directory/Create

public ActionResult Create()

{

return View();

}

// POST: Directory/Create

[HttpPost]

public ActionResult Create(Directory collection)

{

try

{

// TODO: Add insert logic here

dc.Directory.InsertOnSubmit(collection);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: Directory/Edit/5

public ActionResult Edit(string type)

{

var getDirectoryDetails = dc.Directory.Single(x => x.Type == type);

return View(getDirectoryDetails);

}

// POST: Directory/Edit/5

[HttpPost]

public ActionResult Edit(string type, Directory collection)

{

try

{

// TODO: Add update logic here

Directory directory = dc.Directory.Single(x => x.Type == type);

directory.Type = collection.Type;

directory.Notes = collection.Notes;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: Directory/Delete/5

public ActionResult Delete(string type)

{

var getDirectoryDetails = dc.Directory.Single(x => x.Type == type);

return View(getDirectoryDetails);

}

// POST: Directory/Delete/5

[HttpPost]

public ActionResult Delete(string type, Directory collection)

{

try

{

// TODO: Add delete logic here

var directoryDelete = dc.Directory.Single(x => x.Type == type);

dc.Directory.DeleteOnSubmit(directoryDelete);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

**HomeController.cs:**

namespace CadastreManagment.Controllers

{

public class HomeController : Controller

{

public ActionResult Index()

{

return View();

}

public ActionResult About()

{

ViewBag.Message = "Your application description page.";

return View();

}

public ActionResult Contact()

{

ViewBag.Message = "Your contact page.";

return View();

}

public ActionResult Statements()

{

ViewBag.Message = "";

return View();

}

public ActionResult userOffice()

{

ViewBag.Message = "";

return View();

}

}

}

**LandRegistryController.cs:**

namespace CadastreManagment.Controllers

{

public class LandRegistryController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: LandRegistry

public ActionResult Index()

{

//var landRegistryDetails = from x in dc.LandRegistry select x;

//return View(landRegistryDetails);

return View(dc.LandRegistry.ToList());

}

// GET: LandRegistry/Details/5

public ActionResult Details(int id)

{

var getLandRegistryDetails = dc.LandRegistry.Single(x => x.Id == id);

return View(getLandRegistryDetails);

}

[Authorize(Roles = "user")]

public ActionResult IndexId(string userId)

{

List<LandRegistry> landRegistryDetails = new List<LandRegistry>() ;

landRegistryDetails = dc.LandRegistry.Where(x => x.Passport == userId).ToList();

return View(landRegistryDetails);

}

[Authorize(Roles = "admin")]

// GET: LandRegistry/Create

public ActionResult Create()

{

return View();

}

// POST: LandRegistry/Create

[HttpPost]

public ActionResult Create(LandRegistry collection)

{

try

{

// TODO: Add insert logic here

dc.LandRegistry.InsertOnSubmit(collection);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "admin")]

// GET: LandRegistry/Edit/5

public ActionResult Edit(int id)

{

var getLandRegistryDetails = dc.LandRegistry.Single(x => x.Id == id);

return View(getLandRegistryDetails);

}

// POST: LandRegistry/Edit/5

[HttpPost]

public ActionResult Edit(int id, LandRegistry collection)

{

try

{

// TODO: Add update logic here

LandRegistry landRegistry = dc.LandRegistry.Single(x => x.Id == id);

landRegistry.Passport = collection.Passport;

landRegistry.Price = collection.Price;

landRegistry.Address = collection.Address;

landRegistry.ApprovalDate = collection.ApprovalDate;

landRegistry.Area = collection.Area;

landRegistry.Tax\_coefficint = collection.Tax\_coefficint;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: LandRegistry/Delete/5

[Authorize(Roles = "admin")]

public ActionResult Delete(int id)

{

var getLandRegistryDetails = dc.LandRegistry.Single(x => x.Id == id);

return View(getLandRegistryDetails);

}

// POST: LandRegistry/Delete/5

[HttpPost]

public ActionResult Delete(int id, LandRegistry collection)

{

try

{

// TODO: Add delete logic here

var landRegistryDelete = dc.LandRegistry.Single(x => x.Id == id);

dc.LandRegistry.DeleteOnSubmit(landRegistryDelete);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

**MineralsController.cs:**

namespace CadastreManagment.Controllers

{

public class MineralsController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: Minerals

public ActionResult Index()

{

var mineralsDetails = from x in dc.Minerals select x;

return View(mineralsDetails);

}

// GET: Minerals/Details/5

public ActionResult Details(int id)

{

var getMineralsDetails = dc.Minerals.Single(x => x.Id == id);

return View(getMineralsDetails);

}

[Authorize(Roles = "user")]

public ActionResult IndexId(string userId)

{

List<Minerals> mineralsDetails = new List<Minerals>();

mineralsDetails = dc.Minerals.Where(x => x.Passport == userId).ToList();

return View(mineralsDetails);

}

[Authorize(Roles = "admin")]

// GET: Minerals/Create

public ActionResult Create()

{

return View();

}

// POST: Minerals/Create

[HttpPost]

public ActionResult Create(Minerals collection)

{

try

{

// TODO: Add insert logic here

dc.Minerals.InsertOnSubmit(collection);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "admin")]

// GET: Minerals/Edit/5

public ActionResult Edit(int id)

{

var getMineralsDetails = dc.Minerals.Single(x => x.Id == id);

return View(getMineralsDetails);

}

// POST: Minerals/Edit/5

[HttpPost]

public ActionResult Edit(int id, Minerals collection)

{

try

{

// TODO: Add update logic here

Minerals minerals = dc.Minerals.Single(x => x.Id == id);

minerals.Passport = collection.Passport;

minerals.Price = collection.Price;

minerals.Type = collection.Type;

minerals.Tax\_coefficint = collection.Tax\_coefficint;

minerals.Address = collection.Address;

minerals.Amount = collection.Amount;

minerals.Approval\_date = collection.Approval\_date;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "admin")]

// GET: Minerals/Delete/5

public ActionResult Delete(int id)

{

var getMineralsDetails = dc.Minerals.Single(x => x.Id == id);

return View(getMineralsDetails);

}

// POST: Minerals/Delete/5

[HttpPost]

public ActionResult Delete(int id, Minerals collection)

{

try

{

// TODO: Add delete logic here

var mineralsDelete = dc.Minerals.Single(x => x.Id == id);

dc.Minerals.DeleteOnSubmit(mineralsDelete);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

**RealEstateCadastreController.cs:**

namespace CadastreManagment.Controllers

{

public class RealEstateCadastreController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: RealEstateCadastre

public ActionResult Index()

{

var realEstateCadastreDetails = from x in dc.RealEstateCadastre select x;

return View(realEstateCadastreDetails);

}

[Authorize(Roles = "user")]

public ActionResult IndexId(string userId)

{

List<RealEstateCadastre> realEstateCadastreDetails = new List<RealEstateCadastre>();

realEstateCadastreDetails = dc.RealEstateCadastre.Where(x => x.Passport == userId).ToList();

return View(realEstateCadastreDetails);

}

// GET: RealEstateCadastre/Details/5

public ActionResult Details(int id)

{

var getRealEstateCadastreDetails = dc.RealEstateCadastre.Single(x => x.Id == id);

return View(getRealEstateCadastreDetails);

}

[Authorize(Roles = "admin")]

// GET: RealEstateCadastre/Create

public ActionResult Create()

{

return View();

}

// POST: RealEstateCadastre/Create

[HttpPost]

public ActionResult Create(RealEstateCadastre collection)

{

try

{

// TODO: Add insert logic here

dc.RealEstateCadastre.InsertOnSubmit(collection);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "admin")]

// GET: RealEstateCadastre/Edit/5

public ActionResult Edit(int id)

{

var getRealEstateCadastreDetails = dc.RealEstateCadastre.Single(x => x.Id == id);

return View(getRealEstateCadastreDetails);

}

// POST: RealEstateCadastre/Edit/5

[HttpPost]

public ActionResult Edit(int id, RealEstateCadastre collection)

{

try

{

// TODO: Add update logic here

RealEstateCadastre realEstateCadastre = dc.RealEstateCadastre.Single(x => x.Id == id);

realEstateCadastre.Passport = collection.Passport;

realEstateCadastre.Readiness = collection.Readiness;

realEstateCadastre.Type = collection.Type;

realEstateCadastre.Notes = collection.Notes;

realEstateCadastre.Tax\_coefficint = collection.Tax\_coefficint;

realEstateCadastre.Area = collection.Area;

realEstateCadastre.Address = collection.Address;

realEstateCadastre.Appointment = collection.Appointment;

realEstateCadastre.Floors = collection.Floors;

realEstateCadastre.Approval\_Date = collection.Approval\_Date;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "admin")]

// GET: RealEstateCadastre/Delete/5

public ActionResult Delete(int id)

{

var getRealEstateCadastreDetails = dc.RealEstateCadastre.Single(x => x.Id == id);

return View(getRealEstateCadastreDetails);

}

// POST: RealEstateCadastre/Delete/5

[HttpPost]

public ActionResult Delete(int id, RealEstateCadastre collection)

{

try

{

// TODO: Add delete logic here

var realEstateCadastre = dc.RealEstateCadastre.Single(x => x.Id == id);

dc.RealEstateCadastre.DeleteOnSubmit(realEstateCadastre);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

**StatementLandController.cs:**

namespace CadastreManagment.Controllers

{

public class StatementLandController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: StatementLand

public ActionResult Index()

{

List<StatementLand> statementDetails = new List<StatementLand>();

statementDetails = dc.StatementLand.Where(x => x.Status == "Обрабатывается").ToList();

return View(statementDetails);

}

// GET: StatementLand/Details/5

public ActionResult Details(int id)

{

var statementDetails = dc.StatementLand.Single(x => x.Id == id);

return View(statementDetails);

}

public ActionResult IndexId(string userId)

{

List<StatementLand> statementLandDetails = new List<StatementLand>();

statementLandDetails = dc.StatementLand.Where(x => x.Passport == userId).ToList();

return View(statementLandDetails);

}

// GET: StatementLand/Create

public ActionResult Create()

{

return View();

}

// POST: StatementLand/Create

[HttpPost]

public ActionResult Create(StatementLand collection)

{

try

{

// TODO: Add insert logic here

collection.Status = "Обрабатывается";

dc.StatementLand.InsertOnSubmit(collection);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

public ActionResult CreateLand(int id, StatementLand collection)

{

// TODO: Add insert logic here

LandRegistry statementLand = new LandRegistry();

StatementLand statement = dc.StatementLand.Single(x => x.Id == id);

statementLand.Area = statement.Area;

statementLand.Address = statement.Address;

statementLand.Price = statement.Price;

statementLand.ApprovalDate = DateTime.Today;

statementLand.Passport = statement.Passport;

const double V = 0.2;

statementLand.Tax\_coefficint =(float) V;

dc.LandRegistry.InsertOnSubmit(statementLand);

statement.Status = "Одобрено";

dc.SubmitChanges();

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

}

public ActionResult RefusalLand(int id, StatementLand collection)

{

// TODO: Add insert logic here

StatementLand statement = dc.StatementLand.Single(x => x.Id == id);

statement.Status = "Отказано";

dc.SubmitChanges();

return RedirectToAction("Index");

}

// GET: StatementLand/Edit/5

public ActionResult Edit(int id)

{

var getStatementDetails = dc.StatementLand.Single(x => x.Id == id);

return View(getStatementDetails);

}

// POST: StatementLand/Edit/5

[HttpPost]

public ActionResult Edit(int id, StatementLand collection)

{

try

{

// TODO: Add update logic here

StatementLand statementLand = dc.StatementLand.Single(x => x.Id == id);

statementLand.Status = collection.Status;

statementLand.Area = collection.Area;

statementLand.Address = collection.Address;

statementLand.Price = collection.Price;

statementLand.Passport = collection.Passport;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: StatementLand/Delete/5

public ActionResult Delete(int id)

{

var getStatementDetails = dc.StatementLand.Single(x => x.Id == id);

return View(getStatementDetails);

}

// POST: StatementLand/Delete/5

[HttpPost]

public ActionResult Delete(int id, StatementLand collection)

{

try

{

// TODO: Add delete logic here

var statementDelete = dc.StatementLand.Single(x => x.Id == id);

dc.StatementLand.DeleteOnSubmit(statementDelete);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

**StatementMineralsController.cs:**

namespace CadastreManagment.Controllers

{

public class StatementMineralsController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: StatementMinerals

public ActionResult Index()

{

var statementDetails = from x in dc.StatementMinerals select x;

return View(statementDetails);

}

// GET: StatementMinerals/Details/5

public ActionResult Details(int id)

{

var statementDetails = dc.StatementMinerals.Single(x => x.Id == id);

return View(statementDetails);

}

public ActionResult IndexId(string userId)

{

List<StatementMinerals> statementMineralsDetails = new List<StatementMinerals>();

statementMineralsDetails = dc.StatementMinerals.Where(x => x.Passport == userId).ToList();

return View(statementMineralsDetails);

}

// GET: StatementMinerals/Create

public ActionResult Create()

{

return View();

}

// POST: StatementMinerals/Create

[HttpPost]

public ActionResult Create(StatementMinerals collection)

{

try

{

// TODO: Add insert logic here

collection.Status = "Обрабатывается";

dc.StatementMinerals.InsertOnSubmit(collection);

dc.SubmitChanges();

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

}

catch

{

return View();

}

}

public ActionResult CreateLand(int id, StatementMinerals collection)

{

// TODO: Add insert logic here

Minerals minerals = new Minerals();

StatementMinerals statement = dc.StatementMinerals.Single(x => x.Id == id);

minerals.Type = statement.Type;

minerals.Address = statement.Address;

minerals.Price = statement.Price;

minerals.Approval\_date = DateTime.Today;

minerals.Amount = statement.Amount;

minerals.Passport = statement.Passport;

const double V = 0.2;

minerals.Tax\_coefficint = (float)V;

dc.Minerals.InsertOnSubmit(minerals);

statement.Status = "Одобрено";

dc.SubmitChanges();

return RedirectToAction("Index");

}

public ActionResult RefusalLand(int id, Minerals collection)

{

// TODO: Add insert logic here

StatementMinerals statement = dc.StatementMinerals.Single(x => x.Id == id);

statement.Status = "Отказано";

dc.SubmitChanges();

return RedirectToAction("Index");

}

// GET: StatementMinerals/Edit/5

public ActionResult Edit(int id)

{

var getStatementDetails = dc.StatementMinerals.Single(x => x.Id == id);

return View(getStatementDetails);

}

// POST: StatementMinerals/Edit/5

[HttpPost]

public ActionResult Edit(int id, StatementMinerals collection)

{

try

{

// TODO: Add update logic here

StatementMinerals statement = dc.StatementMinerals.Single(x => x.Id == id);

statement.Status = collection.Status;

statement.Type = collection.Type;

statement.Address = collection.Address;

statement.Passport = collection.Passport;

statement.Price = collection.Price;

statement.Amount = collection.Amount;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: StatementMinerals/Delete/5

public ActionResult Delete(int id)

{

var getStatementDetails = dc.StatementMinerals.Single(x => x.Id == id);

return View(getStatementDetails);

}

// POST: StatementMinerals/Delete/5

[HttpPost]

public ActionResult Delete(int id, StatementMinerals collection)

{

try

{

// TODO: Add delete logic here

var statementDelete = dc.StatementMinerals.Single(x => x.Id == id);

dc.StatementMinerals.DeleteOnSubmit(statementDelete);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

**StatementRealEstateController.cs:**

namespace CadastreManagment.Controllers

{

public class StatementRealEstateController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: StatementRealEstate

public ActionResult Index()

{

var statementDetails = from x in dc.StatementRealEstate select x;

return View(statementDetails);

}

// GET: StatementRealEstate/Details/5

public ActionResult Details(int id)

{

var statementDetails = dc.StatementRealEstate.Single(x => x.Id == id);

return View(statementDetails);

}

public ActionResult IndexId(string userId)

{

List<StatementRealEstate> statementRealEstateDetails = new List<StatementRealEstate>();

statementRealEstateDetails = dc.StatementRealEstate.Where(x => x.Passport == userId).ToList();

return View(statementRealEstateDetails);

}

// GET: StatementRealEstate/Create

public ActionResult Create()

{

return View();

}

// POST: StatementRealEstate/Create

[HttpPost]

public ActionResult Create(StatementRealEstate collection)

{

try

{

// TODO: Add insert logic here

collection.Status = "Обрабатывается";

dc.StatementRealEstate.InsertOnSubmit(collection);

dc.SubmitChanges();

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

}

catch

{

return View();

}

}

public ActionResult CreateRealEstate(int id, StatementRealEstate collection)

{

// TODO: Add insert logic here

RealEstateCadastre statementRealEstate = new RealEstateCadastre();

StatementRealEstate statement = dc.StatementRealEstate.Single(x => x.Id == id);

statementRealEstate.Area = statement.Area;

statementRealEstate.Address = statement.Address;

statementRealEstate.Approval\_Date = DateTime.Today;

statementRealEstate.Passport = statement.Passport;

statementRealEstate.Type = statement.Type;

statementRealEstate.Readiness = statement.Readiness;

statementRealEstate.Appointment = statement.Appointment;

statementRealEstate.Floors = statement.Floors;

const double V = 0.2;

statementRealEstate.Tax\_coefficint = (float)V;

dc.RealEstateCadastre.InsertOnSubmit(statementRealEstate);

statement.Status = "Одобрено";

dc.SubmitChanges();

return RedirectToAction("Index");

}

public ActionResult RefuseRealEstate(int id, StatementRealEstate collection)

{

// TODO: Add insert logic here

StatementRealEstate statement = dc.StatementRealEstate.Single(x => x.Id == id);

statement.Status = "Отказано";

dc.SubmitChanges();

return RedirectToAction("Index");

}

// GET: StatementRealEstate/Edit/5

public ActionResult Edit(int id)

{

var getStatementDetails = dc.StatementRealEstate.Single(x => x.Id == id);

return View(getStatementDetails);

}

// POST: StatementRealEstate/Edit/5

[HttpPost]

public ActionResult Edit(int id, StatementRealEstate collection)

{

try

{

// TODO: Add update logic here

StatementRealEstate statement = dc.StatementRealEstate.Single(x => x.Id == id);

statement.Status = collection.Status;

statement.Type = collection.Type;

statement.Area = collection.Area;

statement.Address = collection.Address;

statement.Passport = collection.Passport;

statement.Readiness = collection.Readiness;

statement.Appointment = collection.Appointment;

statement.Floors = collection.Floors;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: StatementRealEstate/Delete/5

public ActionResult Delete(int id)

{

var getStatementDetails = dc.StatementRealEstate.Single(x => x.Id == id);

return View(getStatementDetails);

}

// POST: StatementRealEstate/Delete/5

[HttpPost]

public ActionResult Delete(int id, FormCollection collection)

{

try

{

// TODO: Add delete logic here

var statementDelete = dc.StatementRealEstate.Single(x => x.Id == id);

dc.StatementRealEstate.DeleteOnSubmit(statementDelete);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

**StatementWaterController.cs:**

namespace CadastreManagment.Controllers

{

public class StatementWaterController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: StatementWater

public ActionResult Index()

{

var statementDetails = from x in dc.StatementWater select x;

return View(statementDetails);

}

// GET: StatementWater/Details/5

public ActionResult Details(int id)

{

var statementDetails = dc.StatementWater.Single(x => x.Id == id);

return View(statementDetails);

}

public ActionResult IndexId(string userId)

{

List<StatementWater> statementWaterDetails = new List<StatementWater>();

statementWaterDetails = dc.StatementWater.Where(x => x.Passport == userId).ToList();

return View(statementWaterDetails);

}

// GET: StatementWater/Create

public ActionResult Create()

{

return View();

}

// POST: StatementWater/Create

[HttpPost]

public ActionResult Create(StatementWater collection)

{

try

{

// TODO: Add insert logic here

collection.Status = "Обрабатывается";

dc.StatementWater.InsertOnSubmit(collection);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

public ActionResult CreateWater(int id, StatementWater collection)

{

// TODO: Add insert logic here

WaterCadastre statementWater= new WaterCadastre();

StatementWater statement = dc.StatementWater.Single(x => x.Id == id);

statementWater.Area = statement.Area;

statementWater.Address = statement.Address;

statementWater.Type = statement.Type;

statementWater.Approval\_date = DateTime.Today;

statementWater.Passport = statement.Passport;

statementWater.Water\_management\_facilities = statement.Water\_management\_facilities;

statementWater.Water\_quality = statement.Water\_quality;

const double V = 0.2;

statementWater.Tax\_coefficint = (float)V;

dc.WaterCadastre.InsertOnSubmit(statementWater);

statement.Status = "Одобрено";

dc.SubmitChanges();

return RedirectToAction("Index");

}

public ActionResult RefusalWater(int id, StatementWater collection)

{

// TODO: Add insert logic here

StatementWater statement = dc.StatementWater.Single(x => x.Id == id);

statement.Status = "Отказано";

dc.SubmitChanges();

return RedirectToAction("Index");

}

// GET: StatementWater/Edit/5

public ActionResult Edit(int id)

{

var getStatementDetails = dc.StatementWater.Single(x => x.Id == id);

return View(getStatementDetails);

}

// POST: StatementWater/Edit/5

[HttpPost]

public ActionResult Edit(int id, StatementWater collection)

{

try

{

// TODO: Add update logic here

StatementWater statement = dc.StatementWater.Single(x => x.Id == id);

statement.Status = collection.Status;

statement.Type = collection.Type;

statement.Area = collection.Area;

statement.Address = collection.Address;

statement.Passport = collection.Passport;

statement.Water\_quality = collection.Water\_quality;

statement.Water\_management\_facilities = collection.Water\_management\_facilities;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

// GET: StatementWater/Delete/5

public ActionResult Delete(int id)

{

var getStatementDetails = dc.StatementWater.Single(x => x.Id == id);

return View(getStatementDetails);

}

// POST: StatementWater/Delete/5

[HttpPost]

public ActionResult Delete(int id, StatementWater collection)

{

try

{

// TODO: Add delete logic here

var statementDelete = dc.StatementWater.Single(x => x.Id == id);

dc.StatementWater.DeleteOnSubmit(statementDelete);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

}

}

**TaxController.cs:**

namespace CadastreManagment.Controllers

{

public class TaxController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: Tax

public ActionResult Index()

{

var taxDetails = from x in dc.Tax select x;

return View(taxDetails);

}

[Authorize(Roles = "user")]

public ActionResult IndexId(int cadastreId, string cadastreType)

{

List<Tax> taxDetails = new List<Tax>();

taxDetails = dc.Tax.Where(x => x.Cadastre\_id == cadastreId && x.Cadastre\_type == cadastreType).ToList();

return View(taxDetails);

}

// GET: Tax/Details/5

public ActionResult Details(int id)

{

var getTaxDetails = dc.Tax.Single(x => x.Id == id);

return View(getTaxDetails);

}

[Authorize(Roles = "admin")]

// GET: Tax/Create

public ActionResult Create()

{

return View();

}

// POST: Tax/Create

[HttpPost]

public ActionResult Create(Tax collection)

{

try

{

// TODO: Add insert logic here

dc.Tax.InsertOnSubmit(collection);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "admin")]

// GET: Tax/Edit/5

public ActionResult Edit(int id)

{

var getTaxDetails = dc.Tax.Single(x => x.Id == id);

return View(getTaxDetails);

}

// POST: Tax/Edit/5

[HttpPost]

public ActionResult Edit(int id, Tax collection)

{

try

{

// TODO: Add update logic here

Tax tax = dc.Tax.Single(x => x.Id == id);

tax.Price = collection.Price;

tax.Date\_of\_creation = collection.Date\_of\_creation;

tax.Payment\_date = collection.Payment\_date;

tax.Cadastre\_type = collection.Cadastre\_type;

tax.Cadastre\_id = collection.Cadastre\_id;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "admin")]

// GET: Tax/Delete/5

public ActionResult Delete(int id)

{

var getTaxDetails = dc.Tax.Single(x => x.Id == id);

return View(getTaxDetails);

}

// POST: Tax/Delete/5

[HttpPost]

public ActionResult Delete(int id, Tax collection)

{

try

{

// TODO: Add delete logic here

var taxDelete = dc.Tax.Single(x => x.Id == id);

dc.Tax.DeleteOnSubmit(taxDelete);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "user")]

public ActionResult EditPaymentDate(int id)

{

var getTaxDetails = dc.Tax.Single(x => x.Id == id);

return View(getTaxDetails);

}

// POST: Tax/Edit/5

[HttpPost]

public ActionResult EditPaymentDate(int id, Tax collection)

{

try

{

// TODO: Add update logic here

Tax tax = dc.Tax.Single(x => x.Id == id);

tax.Price = collection.Price;

tax.Date\_of\_creation = collection.Date\_of\_creation;

tax.Payment\_date = DateTime.Today;

tax.Cadastre\_type = collection.Cadastre\_type;

tax.Cadastre\_id = collection.Cadastre\_id;

dc.SubmitChanges();

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

}

catch

{

return View();

}

}

}

}

**WaterCadastreController.cs:**

namespace CadastreManagment.Controllers

{

public class WaterCadastreController : Controller

{

DataClasses1DataContext dc = new DataClasses1DataContext();

// GET: WaterCadastre

public ActionResult Index()

{

var waterCadastreDetails = from x in dc.WaterCadastre select x;

return View(waterCadastreDetails);

}

// GET: WaterCadastre/Details/5

public ActionResult Details(int id)

{

var getWaterCadastreDetails = dc.WaterCadastre.Single(x => x.Id == id);

return View(getWaterCadastreDetails);

}

[Authorize(Roles = "user")]

public ActionResult IndexId(string userId)

{

List<WaterCadastre> waterDetails = new List<WaterCadastre>();

waterDetails = dc.WaterCadastre.Where(x => x.Passport == userId).ToList();

return View(waterDetails);

}

[Authorize(Roles = "admin")]

// GET: WaterCadastre/Create

public ActionResult Create()

{

return View();

}

// POST: WaterCadastre/Create

[HttpPost]

public ActionResult Create(WaterCadastre collection)

{

try

{

// TODO: Add insert logic here

dc.WaterCadastre.InsertOnSubmit(collection);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "admin")]

// GET: WaterCadastre/Edit/5

public ActionResult Edit(int id)

{

var getWaterCadastreDetails = dc.WaterCadastre.Single(x => x.Id == id);

return View(getWaterCadastreDetails);

}

// POST: WaterCadastre/Edit/5

[HttpPost]

public ActionResult Edit(int id, WaterCadastre collection)

{

try

{

// TODO: Add update logic here

WaterCadastre waterCadastre = dc.WaterCadastre.Single(x => x.Id == id);

waterCadastre.Passport = collection.Passport;

waterCadastre.Type = collection.Type;

waterCadastre.Area = collection.Area;

waterCadastre.Address = collection.Address;

waterCadastre.Approval\_date = collection.Approval\_date;

waterCadastre.Tax\_coefficint = collection.Tax\_coefficint;

waterCadastre.Water\_management\_facilities = collection.Water\_management\_facilities;

waterCadastre.Water\_quality = collection.Water\_quality;

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

}

[Authorize(Roles = "admin")]

// GET: WaterCadastre/Delete/5

public ActionResult Delete(int id)

{

var getWaterCadastreDetails = dc.WaterCadastre.Single(x => x.Id == id);

return View(getWaterCadastreDetails);

}

// POST: WaterCadastre/Delete/5

[HttpPost]

public ActionResult Delete(int id, WaterCadastre collection)

{

try

{

// TODO: Add delete logic here

var waterCadastre = dc.WaterCadastre.Single(x => x.Id == id);

dc.WaterCadastre.DeleteOnSubmit(waterCadastre);

dc.SubmitChanges();

return RedirectToAction("Index");

}

catch

{

return View();

}

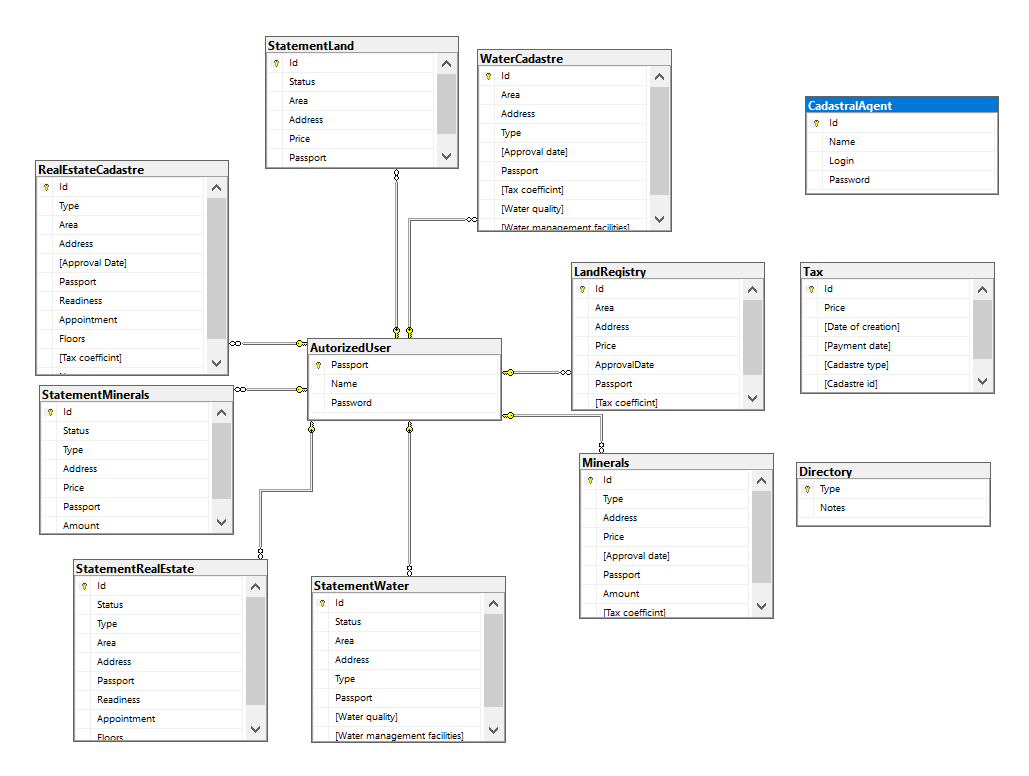
}

}

}

## ПРИЛОЖЕНИЕ Б

## СТРУКТУРА ТАБЛИЦЫ БД

****