Софийски университет „Св. Климент Охридски”
Факултет по Математика и Информатика

ДИПЛОМНА РАБОТА

Тема: *Управление на процеса на актуване на държавни и общински имоти в Република България.*

Разработил:

Стефан Христов Захариев
Ф.Н. М21904

Научен ръководител:

доц. Б. Бончев

гр. София
2007 г.

Съдържание

Индекс на фигурите

1 Въведение

1.1 Увод

Развитието на информационните технологии все-повече намалява тяхната себестойност спрямо производителността, която те предлагат. Именно това допринася за все по-широкото им навлизане в нашето ежедневие. Една от областите, където те намират най-широко приложение е документо-обработката, заради по-добрата възможност за търсене, по-високата надеждност и т.н.

Типична такава област, за която воденето на документацията е изключително важно е сферата на недвижимите имоти. Настоящата дипломна работа разглежда именно процеса на автоматизация обработката на общинско-областните имоти.

1.2 Цел на работата

Цел на настоящата дипломна работа е разработката на модул за управление на процеса на актуване на държавни и общински имоти в Република България. Модулът за актуване представлява част от програмното приложение „Общинско-областна система за имоти”. Това програмно приложение е част от „Информационната система за управление на държавни и общински имоти (ИСУДОИ)”. ИСУДОИ представлява пакет от приложения за управление на документи и регистри за кадастрална информация (данни за физическите характеристики на даден имот – граници, площ, вид и др.), както и на данни, поддържащи процеса на актуване на имотите и на съхраняване на издадените актове.

При проектирането на системата са следвани две основни изисквания:

- да е ориентирана към законовите разпоредби в Република България;

- да отразява адекватно съществуващите процеси на обмен, обработка, управление и съхранение на информацията към момента на разработката.

1.3 Структура

Първата глава на дипломната работа прави въведение в разработката. Тя описва целите поставени пред нея, както структура и. Дадена е кратка таблица с използваните съкращения.

Във втората глава се описват изискванията поставени пред разработваната система. Посочена е и подобна разработка направена от Siemens.

Третата глава описва използваните технологии и инструменти. На кратко са описани най-важните от тях като .NET Framework, SQL Server и др.

Следващата, четвърта глава съдържа проектирането и имплементацията на системата. Дадени са случаите на употреба, архитектурата, базата данни, потребителския интерфейс, клас, компонент и диаграмите на последователността.

В пета глава се разглежда тестването и внедряването на системата. Описва се резултата от внедряването в Министерски съвет на Република България.

Глава шеста прави заключение за разработката на системата. Разглеждат се също така и насоките за бъдещо развитие на системата.

В последната седма глава е поместена използваната литература по време на разработката.

Към дипломната работа има четири приложения. Първото от тях съдържа Ръководство на потребителя за работата му с модул актуване.

Във второто приложение се съдържа изходния код на някой от по-важните класове, които изграждат настоящата система.

Подготовката на работно място за разработка е разгледано в приложение номер три, а приложение четири съдържа използваните конвенции за изходния код ползвани при разработката.

1.4 Използвани съкращения

Съкращение	Пълно наименование
ОСИ	Общинско-областна система за имоти
ИР	Имотен регистър
ИСУДОИ	Информационна система за управление на държавни и общински имоти
CLR	Common Language Runtime
E-R	Entity-Relationship
FCL	Framework Class Library
HQL	Hibernate Query Language
IDE	Integrated Development Environment
SQL	Structured Query Language
UML	Unified Modeling Language
VP	Visual Paradigm
WSE	Web Service Enhancements
XML	eXtensible Markup Language

2 Проблемна област и съществуващи системи

Модул „Актуване" поддържа актуален регистър на актовете за държавна/общинска собственост, както и автоматизиране на процеса по издаване на нов акт или корекция на съществуващ. Формулярите (актовете) се отпечатват съобразно ВСИЧКИ нормативни изисквания и се прикрепят към записа за акт (регистъра) под формата на сканирани документи.

Процесът по актуване на държавните/общински имоти започва със създаването на запис в базата данни за съответният акт, като същевременно потребителят прикрепва към записа нормативен документ – скица, устройствен план и/или друг, на база на който се извършва актуването. Този документ е предварително сканиран и обхванат в базата данни чрез модул „Дигитализиране на документи".

Особеност при актуването е, че с един акт могат да бъдат актувани един или повече имоти (на база законодателството), тогава, когато за същите имоти има само една партида в Имотния регистър.

След записване на въведените данни ръководителя на отдела получава досието на съответния акт и проверява на екран дали всички атрибути са въведени коректно. При наличие на грешка/и/ записва забележка, отказва да одобри акта (поставя го в състояние Отказан от формален контрол) и го връща за корекция на Оперативния служител. Когато всички въведени данни са коректни и пълни Ръководителя на отдел одобрява акта (поставя го в състояние Одобрен формален контрол), като въвежда кмета или областния управител (оправомощеното длъжностно лице), който следва да подпише и утвърди акта, и разпечатва формуляра за утвърждаване (прикрепя го към досието на акта).

Формално одобрените актове подлежат на пълен преглед от оправомощеното лице, което ги утвърждава (обикновено кмет на населено място или областен управител), като при наличие на основание за отказ – то се отразява в записа под формата на забележка и кмета/об. управител отхвърля акта за нови корекции/изтриване (поставя го в състояние Отхвърлен от утвърждаване). В този случай целият процес се стартира отново. Когато всички предпоставки са налице, кмета/областният управител утвърждава акта и същевременно подписва оригиналния хартиен документ (поставя акта в състояние Утвърден). Тази процедура автоматично генерира Заявка за вписване в Имотния регистър, която подлежи на одобряване (реално вписване) от съдия по вписванията.

При отказ от съдията по вписванията процедурата стартира отначало (актът е поставен в състояние Отказано вписване и подлежи на корекция/изтриване).

При извършване на вписване, актът се поставя в състояние Вписан, след което чрез модул „Дигитализиране на документ" се създава (сканира и обхваща) документ „Вписан акт", който се прикрепя към записа за акт. С това актът е финализиран (състояние Приключен) и не подлежи на последващи корекции.

Описания по-горе процес на работа е базиран на добрите практики установени в провинция Бавария, Германия. На базата на тези добри практики е създадена и системата на Siemens - SolumSTAR http://www.solumstar.nrw.de/. Тя позволява текстова автоматизация, архивиране, търсене/проучване и електронно водене на регистъра с имотите.

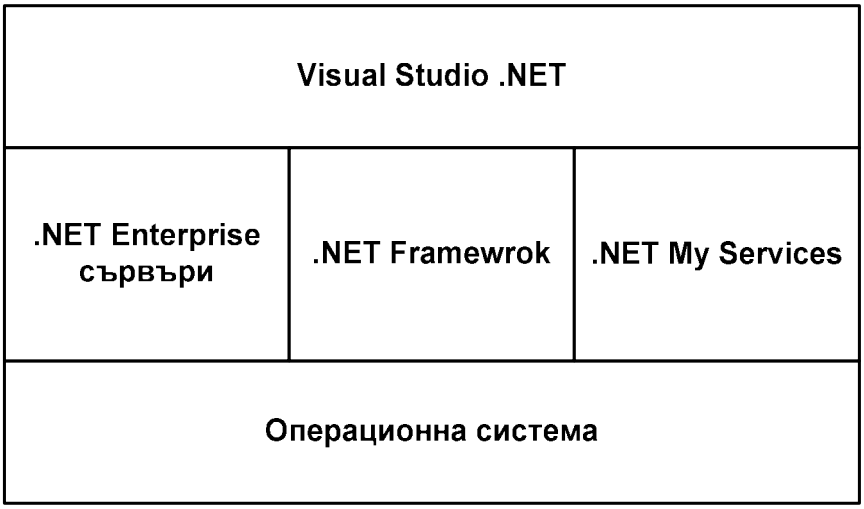
3 Използвани технологии и инструменти

Настоящата система е разработена на база на едни от последните технологии, предлагани от Microsoft. Изключение прави само NHibernate, който е проект с отворен код и се разработва от независими разработчици.

3.1 Microsoft .NET Framework

През юли месец 2000 година, Microsoft обявяват инициативата си за създаване на .NET платформата. Тя представлява нова среда за разработка на приложения, която идва с концепции, непознати дотогава в средствата за разработка на Microsoft. Същевременно се обединяват и технологиите на компанията създадени през 90-те години.

Ново обявената платформа се състои от 5 основни компонента, които са показани на фиг. 1



фиг. 1 Компоненти на платформата .NET

На най-ниското ниво се намира операционната система, която може да бъде някоя от фамилията операционни системи Windows (Windows 2003, Windows XP и др.).

Едно ниво над операционните системи се намират т.нар. .NET Enterprise сървъри, които спомагат за разработката на големи бизнес приложения. Сред тези сървъри се включват SQL Server (използван в настоящата дипломна разработка), Internet Security and Acceleration Server, Content Management Server, BizTalk Server и др.

От Microsoft разработват и група стандартизирани web услуги (наречени по-късно .NET My Services), които осигуряват решение на често срещани проблеми. Пример за такава услуга е Microsoft Passport, който позволява на потребителите на различни web страници поддържащи споменатата услуга, да ползват едно потребителско име и парола при достъпа им.

Най-отгоре на показаната фигура се намира ново средство за разработка, наречено Visual Studio .NET. То може да се класифицира към така наречените инструменти за бърза разработка (rapid development tools). Новата среда за разработка поддържа различни езици за програмиране, разработка на web услуги, диалогово базирани Windows приложения, както и динамични web страници.

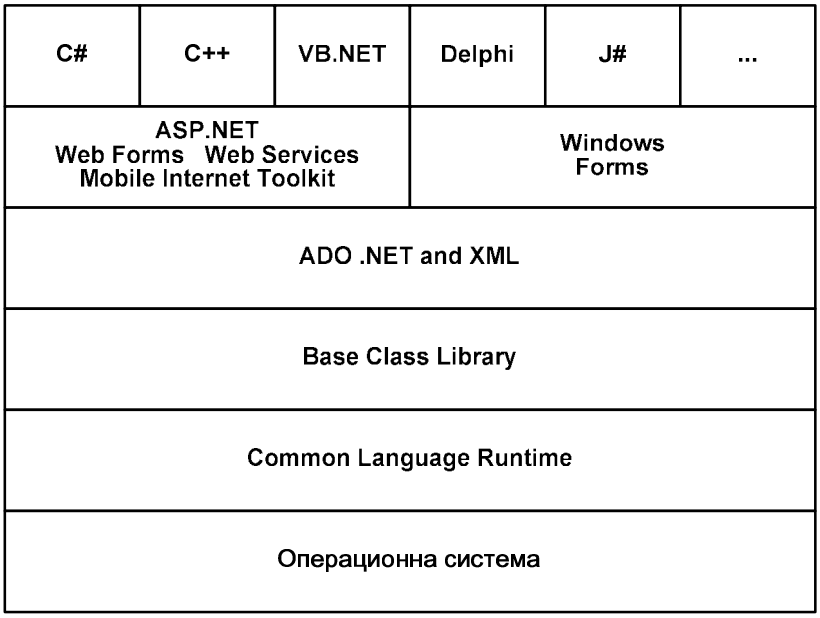
В центъра на .NET платформата се намира .NET Framework – нова среда за разработка и изпълнение на приложения. Тя предлага независимост от използване на езика за програмиране, т.е едно приложение може да бъда написано на няколко различни програмни езици. Освен независимост, се предлага и интеграция между различните езици. Това означава, че разработчикът може да наследява класове, обработва изключения и използва полиморфизъм във взаимодействащи си части на приложението писани на различни езици.

.NET Framework се състои от следните части:

1. **Common Language Runtime (CLR)** – Това е среда, която изпълнява приложенията написани на .NET. Тя също така контролира изпълнението на програмния код. Може да се оприличи на виртуалната машина на Java.

2. **Framework Class Library (FCL)** – Представлява стандартна библиотека от класове спомагаща разработването на .NET приложения. Предоставя основната функционалност за разработка: ADO.NET, XML, ASP.NET, Web Services, Windows Forms.

Архитектурата на .NET Framework е показана на фиг. 2



фиг. 2 Архитектура на .NET Framework

Най-долния слой представлява операционната система. Тя управлява ресурсите, процесите и потребителите на машината. Осигурява на приложенията някои услуги като COM+, MSMQ, IIS. В операционната система CLR представлява отделен процес.

Едно ниво над операционната система се намира CLR. В този слой се извършва управлението на процеса на изпълнение на .NET код. Тук също така се управлява паметта, конкурентността, сигурността на приложенията.

Над CLR се намира Base Class Library. Тя представлява богата обектно-ориентирана библиотека с основни класове предлагаща примитиви за колекции, вход-изход, работа със символни низове, работа с мрежа, сигурност, отдалечено извикване, многонишковост и др.

Работата с данни се подпомага от слоя намиращ се над Base Class Library, а именно библиотеките ADO.NET и XML. Те предлагат класове за достъп до бази данни, реализиране на несвързан (disconnected) модел на данните, силна поддръжка на XML.

Следващият слой предлага класове за реализиране на потребителски интерфейс на приложенията, който може да бъде web базиран или стандартният потребителски интерфейс на Windows. Тук също така се намират и класовете за изграждане на web услуги.

Най-горния слой представляват езиците за програмиране. Както се вижда, за .NET платформата съществуват голям брой програмни езици, като техният брой непрекъснато се увеличава.

Повече информация за Microsoft .NET Framework може да бъде намерена в [1, 7, 10, 15]

3.2 Microsoft Web Service Enhancements 3.0

Web Service Enhancements (WSE) е допълнителна библиотека предлагана от Microsoft към .NET framework, която позволява създаването на уеб услуги базирани на последните протоколи за уеб услуги, като WS-Security, WS-Routing, WS-Attachments. WSE се интегрира напълно с ASP.NET уеб услугите и се явява тяхно разширение, а не заместник.

Ползването на WSE добавя възможност за реализиране на следните 3 ключови характеристики:

1) повишаване на стандартната защита на уеб услугите, която се предлага от ASP.NET
2) SOAP съобщенията предавани между отделни уеб услуги може да бъдат прозрачно рутирани
3) Комуникацията между уеб услугите може да съдържа данни, които не са сериализирани в XML формат

3.3 NHibernate

NHibernate е библиотека, която е типичен представител на т. нар. ORM инструменти. Тя запазва и чете обекти на C# в и от релационна база от данни. За да извърши тези операции върху обектите, програмистът трябва да създаде един xml файл, в който се описва съответствието между класа на C# и таблицата в базата данни. Сред ключовите характеристики, предлагани от библиотеката са:

- естествена поддръжка на принципи от обектно-ориентираното програмиране, като наследяване, полиморфизъм и т.н. За да се запази един клас, който използва наследяване в база от данни не са необходими каквито и да е промени по него.
- поддръжка на т.нар. HQL (Hibernate Query Language). Това е език, който наподобява SQL, но вместо за извличане на записи от релационна база от данни се използва за извличане на обекти на C#.
- разнообразни начини за съхранение на обектите на C# в базата от данни. NHibernate предлага възможност за запазване на всеки клас в собствена таблица, всички класове може да се запазват в една таблица или базов клас се запазва в една таблица, а данните от дъщерен клас се запазват в друга таблица.
- автоматично зареждане на цял граф от обекти. При зареждане на обект от базата от данни е възможно да се заредят автоматично и всички обекти, които текущо зареждания обект реферира.
- поддръжка на т.нар. зареждане при нужда. Ако един обект поддържа колекция от обекти, то зареждането на колекцията може да стане при първоначално използване на елемент колекцията, а не при зареждане на притежаващият я обект. Всичко това се извършва от NHibernate, по такъв начин, че цялата операция е прозрачна за приложния програмист.

- автоматично генериране на първични ключове. За всяка таблица от една релационна база от данни се дефинира първичен ключ. Тъй като обектите на C# се записват в такава таблица, то е наложително в процеса на записа да се генерира първичен ключ. NHibernate предлага няколко алтернативи за решаване на този проблем – използване на предлаганите от релационната база от данни средства, генериране на идентификатори от програмиста и т.н.
- поддръжка на голям набор на бази от данни. NHibernate има възможност за работа с голям брой бази от данни – Oracle, Microsoft SQL Server, Firebird, PostgreSQL и др. Сред тях са както комерсиални продукти, така и такива с отворен код.
- генериране на оптимизиран и сигурен SQL код, който се изпълнява на сървъра за управление на релационната база от данни. По този начин се избягват проблеми, като пробив в сигурността, дължащ се на атаки от вида инжектиране на SQL код (SQL code injection).
- вградена поддръжка на нововъведените в C# generic и nullable типове.

3.4 Microsoft SQL Server 2005

Microsoft SQL Server е система за управление на релационни бази от данни, разработвана от Microsoft. Характерно за него е лесната администрация, подобно на останалите сървъри на Microsoft, добрата производителност и увеличаващата се с всяка следваща версия надеждност. За съжаление Microsoft SQL Server работи само под операционната система Windows.

В момента актуалната версия на продукта е 2005, като в скоро време предстои да излезе версия 2008.

Някой от ключовите характеристики, които притежава този сървър са:

- поддръжка на надмножество на стандартния език SQL, наречено T-SQL. То обогатява езика, като включва в него логически конструкции, цикли и други характерни за езиците от високо ниво конструкции.
- администрацията на сървъра се извършва от графични потребителски мениджъри, които я облекчават значително.
- поддръжка на репликации. Репликация се нарича процеса, при който модификацията на запис от база данни на един сървър се копира автоматично на друг сървър.
- поддръжка на разпределени транзакции. Една транзакция може да обхване няколко сървъра, като е гарантирано, че при успешното и приключване, тя ще приключи успешно на всеки един от сървърите.
- инструменти за прехвърляне на информация от различни формати и релационни бази от данни към Microsoft SQL Server – Data Transformation Services.
- възможност за автоматично превключване към поддържащ сървър в случай на срив на основния.
- индексирани изгледи.
- множество средства за скалиране на базата от данни – поддръжка на разпределени изгледи, състоящи се от таблици разположени върху множество сървъри.
- инструменти за аналитична обработка на данни.

3.4.1 Версии на SQL Server

- **SQL Server 2005 Enterprise Edition** - това е най-пълната версия на продукта, предназначена да работи в банки, правителствени институции и медицински организации. Тази версия е оптимизирана за висока надеждност и разширяемост.
- **SQL Server 2005 Standard Edition** - това е версията предназначена за малкия и средния бизнес. Проектирана е за по-малко натоварване

и не разполага с някой от инструментите за анализ, предлагани от Enterprise версията.

- **SQL Server 2005 Workgroup Edition** – предназначена е за малки организации, които имат нужда от база данни, която да няма ограничение в големината и броя потребители. При нужда лесно може да се премине към Standart версията на продукта.
- **SQL Server 2005 Developer Edition** - това е версията, която е предназначена за разработчици. Тя включва всички възможности на Enterprise версията. Разпространява се под специален режим, който позволява използването и само за разработка и тестване.
- **SQL Server 2005 Express Edition** - това е специална версия, която може да се разпространява от разработчиците на приложения безплатно заедно с техния софтуер. Подари това, не се поддържа пълната функционалност на продукта. Графичните инструменти за администрация не са включени в инсталацията. Те могат да бъдат свалени отделно.

Повече информация за Microsoft SQL Server може да бъде намерена в [2, 8, 9]

3.5 Visual Paradigm for UML

В настоящата разработка се използват диаграми създадени със средата Visual Paradigm for UML (VP-UML). VP-UML представлява мощен инструмент за UML моделиране, предвиден за широк кръг потребители, включително програмисти, системни и бизнес анализатори, системни архитекти, които се интересуват от софтуер за надеждно системно моделиране с използването на обектно-ориентирания подход.

Освен за създаването на UML диаграми, в последните си версии продукта има поддръжка за Entity-Relationship диаграми, диаграми на бизнес процеси и др.

За не комерсиални цели има специална версия (Community Edition) на продукта с ограничени възможности, която е безплатна. Въпреки това, версията която се използва в настоящата разработка е по съвместната академична програма между Visual Paradigm и Факултета по математика и информатика на Софийския университет Св. Климент Охридски.

4 Проектиране и имплементация

Реализацията на софтуерни системи с нарастваща сложност, паралелно от множество екипи от специалисти изисква прилагането на общи стандарти и ефективни методологии за разработка. От появата си през есента на 1995г. до сега Унифицираният език за моделиране – UML се превърна фактически в стандартен „графичен език за визуализиране, специфициране, конструиране и документиране на елементите на една софтуерна система". Именно поради тази причина, настоящата разработка залага именно на UML диаграмите за документация на системата при нейното проектиране и реализация.

Повече информация за UML може да бъде намерена в [4]

4.1 Случай на употреба

Диаграмите на случаите на употреба (Use-case Diagrams) са основен вид диаграми за специфициране и документиране на различните потребителски изисквания към една система. По-точно, те се използват за специфициране на актьорите за системата (потребители и външни системи) и за определяне на функционалните изисквания към разработваната софтуерна система от гледна точка на различните типове потребители и външни клиенти.

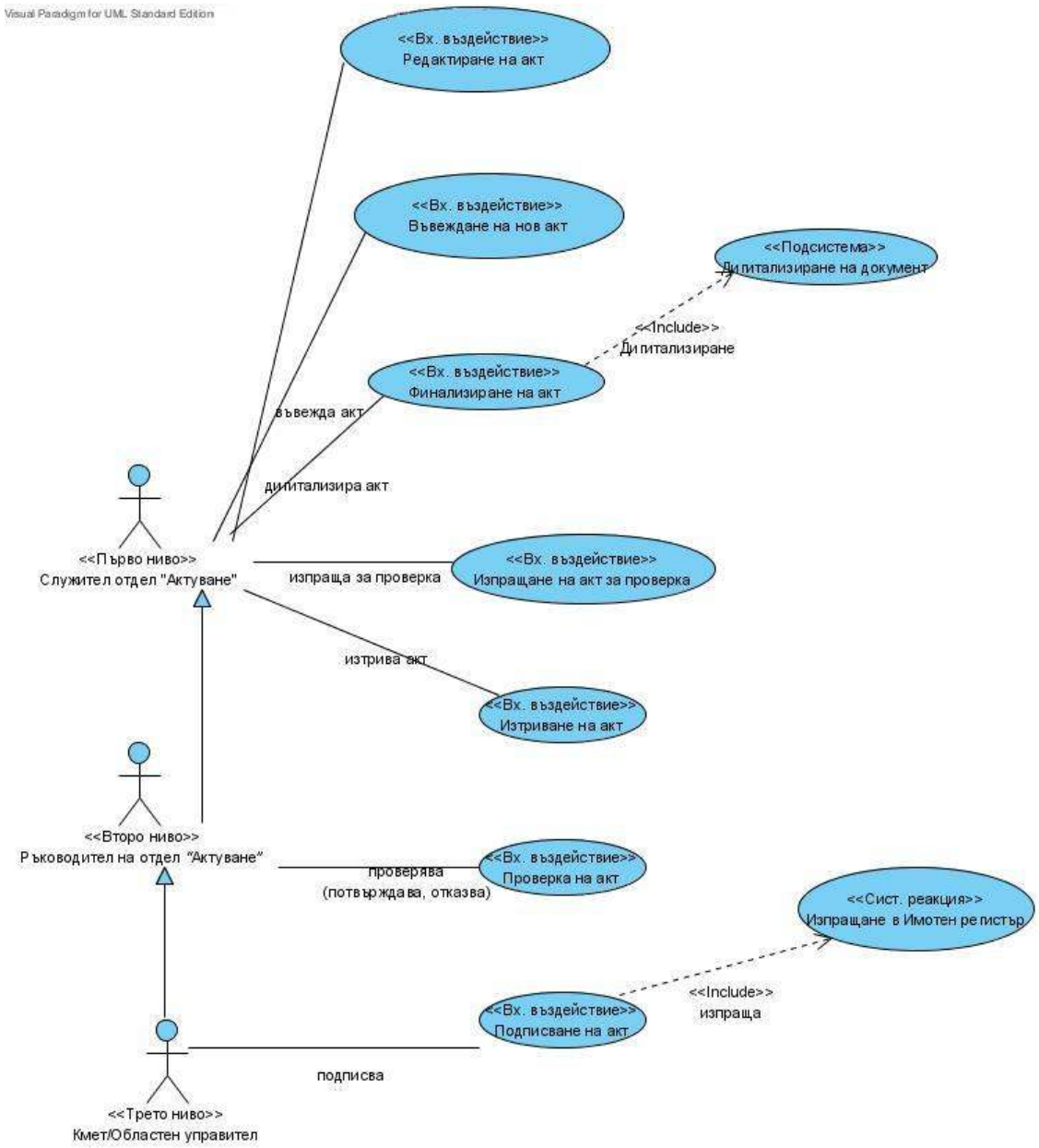
Случаите на употреба могат да имат различни стереотипи, описващи вида на ролята им в цялостния модел и нивата им на отговорност. В изградения модел са използвани следните стереотипи:

- **Нулево ниво** – потребители с този стереотип представляват граждани, правещи публични справки през дадена система (най-вероятно през Уеб интерфейс към ИР)
- **Първо ниво** – потребители с този стереотип въвеждат и редактират данни в системата посредством потребителския интерфейс, а още могат да правят справки за въведени

данни/документи; след приключване на редакцията те генерират чрез системата заявка за потвърждение на въведените данни;

- **Второ ниво –** редактират и потвърждават въведените данни; след приключване на евентуална редакция и потвърждението те генерират чрез системата заявка за утвърждаване (узаконяване) на въведените данни;
- **Трето ниво –** редактират и юридически (крайно) узаконяват документ, както и на хартия, така и в системата - т.е. подписват на хартиен носител и маркират като подписани в системата. След това следва евентуално дигитализиране на утвърдения документ;
- **Четвърто ниво –** такова ниво имат потребителите-администратори на системата; те могат да създават и редактират роли, да създават и редактират системни потребители, и да извършват системни функции.

На следващата фиг. 3 е показана общата диаграмата на случаите на употреба. Всеки случай е описан детайлно в последствие.



фиг. 3 Диаграма на главните случаи на употреба

4.1.1 Описание на актьорите

Име на актьор	Администратор на ИР
Стереотип	Четвърто ниво
Документация	
• създава нова роля в системата, като и присвоява съответни права	

за достъп до ресурсите (правата за твърдо кодирани за системата и биват: достъп по четене, по запис, и по изтриване);

- редактира дадена роля, като й променят правата за достъп до ресурсите;
- регистрира нови потребители на системата с дадена актьорска роля (виж по-долу) и статус (активен/неактивен потребител - потребители не се трият, а само се деактивират!);
- променя ролята и статуса на даден системен потребител;
- извършва системни функции - например четене/запис на външен носител на логове.

Име на актьор	Служител отдел "Актуване"
Стереотип	Първо ниво

Документация

- при наличие на правно основание създава акт за частна или публична собственост
- прикрепва към записа нормативен документ – скица, устройствен план и/или друг, на база на който се извършва актуването
- внася корекции при наличие на грешки
- въвежда към съответния акт данните от документ за вписване на акта, а още и следните действия:
- въвежда данни от документ за вписване към акта, за който се отнасят
- сканира и прикрепя документа за вписване
- със записването във входящия регистър отбелязва в съответната част на партидата на имота, че е постъпила молба за вписване.

Име на актьор	Ръководител на отдел "Актуване"
Стереотип	Второ ниво
Документация	
• проверява/потвърждава или отказва въведени данни за акта за собственост • проверява/потвърждава или отказва данните от документ за вписване на акта (документът се издава от съдия) • в случай на отказ, записва мотивът за отказа в системата • разпечатва формуляра за утвърждаване (прикрепя го към досието на акта) • изпраща съобщение до кмета, който трябва да резолира съставения акт	

Име на актьор	Кмет/Областен управител
Стереотип	Трето ниво
Документация	
• пълен преглед на формално одобрените актове • утвърждава/отказва акт за собственост • при наличие на отказ, той се отразява под формата на забележка • подписва оригиналния хартиен документ	

4.1.2 Описание на случаите на употреба

Име на случай	Въвеждане на нов акт
Кратко описание	В системата се въвежда акт за имот.
Предварителн	Актуването става на имот, който има кадастрален

и условия	номер(партида). При актуване на повече от един имот, те да са в една партида. Имот се актува само веднъж.
Крайни условия	Акта и сканираните документи записани в базата

Основен поток от събития	**Входно въздействие**	**Системна реакция**
	1 Създава нов акт	
	2	Издава номер на акта
	3 Въвежда атрибути на акта	
	4 Скан./дигитализиране (случай-подсистема Дигитализиране на документ)	

Име на случай	Проверка на акт
Кратко описание	Потребител 2-ро ниво проверява дали акта е бил правилно въведен от потребител 1во ниво
Предварителни условия	В системата има въведен акт, който е в състояние "За формален контрол"
Крайни условия	Акта е разгледан от потребител 2ро ниво

Основен поток	**Входно въздействие**	**Системна реакция**
	1 Избор на акт	
	2	Извеждане акта на екрана
	3 Преглед атрибутите на акта за тяхната	

	достоверност	
	4 Ако някой от атрибутите не е коректен - преход към случай Отказ	
	5 Установяване на акта в състояние "Одобрен формален контрол"	
	6 Запис на акта	
Отказ	**Входно въздействие**	**Системна реакция**
	1 Вписване на забележка за обосновка на отказа	
	2 Установяване на акта в състояние "Отказан от формален контрол"	
	3 Запис на акта	

Име на случай	Финализиране на акт	
Кратко описание	В системата се въвежда дигитализирано копие на утвеърдения акт от потребител 3то ниво	
Предварителни условия	Има утвърден на хартия акт	
Крайни условия	Хартиеното копие на утвърдения акт е дигитализирано и въведено в системата	
Поток от	**Входно въздействие**	**Системна реакция**

събития	1	Избор на акт, към който да се прикрепи дигитализирания подписан от кмета документ	
	2		Извеждане акта на екран
	3	Установяване състоянието на акта на Подписан	
	4	Дигитализиране на подписаното хартиено копие	
	5	Запис на акта в ДБ	

Име на случай	Изпращане на акт за проверка		
Кратко описание	Установяване на даден акт в състояние за проверка от потребител 2ро ниво		
Предварителни условия	В системата има въведен акт		
Крайни условия	Акта е в състояние за проверка от потребител 2-ро ниво		
Поток от събития		**Входно въздействие**	**Системна реакция**
	1	Избор на акт	
	2		Визуализиране акта на екран
	3	Установяване състоянието на акта за	

	проверка	
	4 Запис на акта в БД	

Име на случай	Редактиране на акт	
Кратко описание	Редактиране на акт от потребител 1-во ниво	
Предварителни условия	В системата има въведен акт	
Крайни условия	Атрибутите на акта са променени	
Поток от събития	**Входно въздействие**	**Системна реакция**
	1 Избор на акт от потребителя	
	2	Извеждане на акта на екран
	3 Редактиране на атрибутите на акта	
	4 Редактиране (изтриване/добавяне) на прикрепените дигитализирани документи	
	5 Записване на акта в системата	

Име на случай	Подписване на акт
Кратко	Утвърждаване на акт от потребител 3то ниво

описание		
Предварителни условия	В системата има проверен акт от потребител 2ро ниво	
Крайни условия	Акта е утвърден от потребител 3то ниво	
Основен поток	**Входно въздействие**	**Системна реакция**
	1 Избор на акт	
	2	Визуализиране акта на екран
	3 Преглед на акта	
	4 Ако акта е невалиден, преход към случай отказ	
	5 Установяване на акта в състояние "Утвърден"	
Отказ	**Входно въздействие**	**Системна реакция**
	1 Вписване на забележка за обосновка на отказа	
	2 Установяване на акта в състояните "Отхвърлен от утвърждаване"	
	3 Запис на акта	

Име на случай	Редактиране на акт

Кратко описание	Редактиране на акт от потребител 1во ниво		
Предварителни условия	В системата има въведен акт, който не е бил изпращан за проверка или се намира в състояние "Отказан от формален конрол" на ниво 2		
Крайни условия	Атрибутите на акта са променени		
Поток от събития		**Входно въздействие**	**Системна реакция**
	1	Избор на акт	
	2		Визуализиране акта на екран
	3	Редактиране атрибутите на акта	
	4	Записване на акта	

Име на случай	Изтриване на акт от потр. 1-во ниво		
Кратко описание	Изтриване на акт от системата		
Предварителни условия	В системата има въведен акт. Актът не трябва да бъде одобрен за подпис от потребител 2-ро ниво.		
Крайни условия	Актът е изтрит от системата (или маркиран като изтрит).		
Поток от събития		**Входно въздействие**	**Системна реакция**
	1	Избор на акт	
	2		Показване акта на екран
	3	Изтриване на акта	

Име на случай	Изпращане на акт в Имотен регистър	
Кратко описание	Изпращане на акта в системата за Имотен регистър (онлайн)	
Предварителни условия	Има подписан акт от потребител 3-то ниво	
Крайни условия	Акта е изпратен в системата за Имотен регистър	
Основен поток	**Входно въздействие**	**Системна реакция**
	1	Генериране на заявка за вписване в системата имотен регистър
	2	Изпращане на заявката в системата Имотен регистър
	3	Ако изпращането е неуспешно, преход към случай Неуспешно изпращане
Неуспешно изпращане	**Входно въздействие**	**Системна реакция**
	1	Изчакване на определен интервал от време
	2	Преход към Основен поток

4.2 Архитектура на цялостната система

фиг. 4 представя диаграма на общата софтуерна архитектура на системите ОСИ и ИР. Използван е модулен подход, като имената на

модулите, хранилищата, документите и интерфейсите предвидени за бъдещи разширения на разработката, са означени със знака *. Това са следните обекти в диаграмата:

- Модул наеми и аренди
- Хранилище за данни за наеми
- Интерфейс за създаване и редактиране на договори за наем
- Интерфейс за вписвания на актове от нотариуси
- Вписани договори за наем

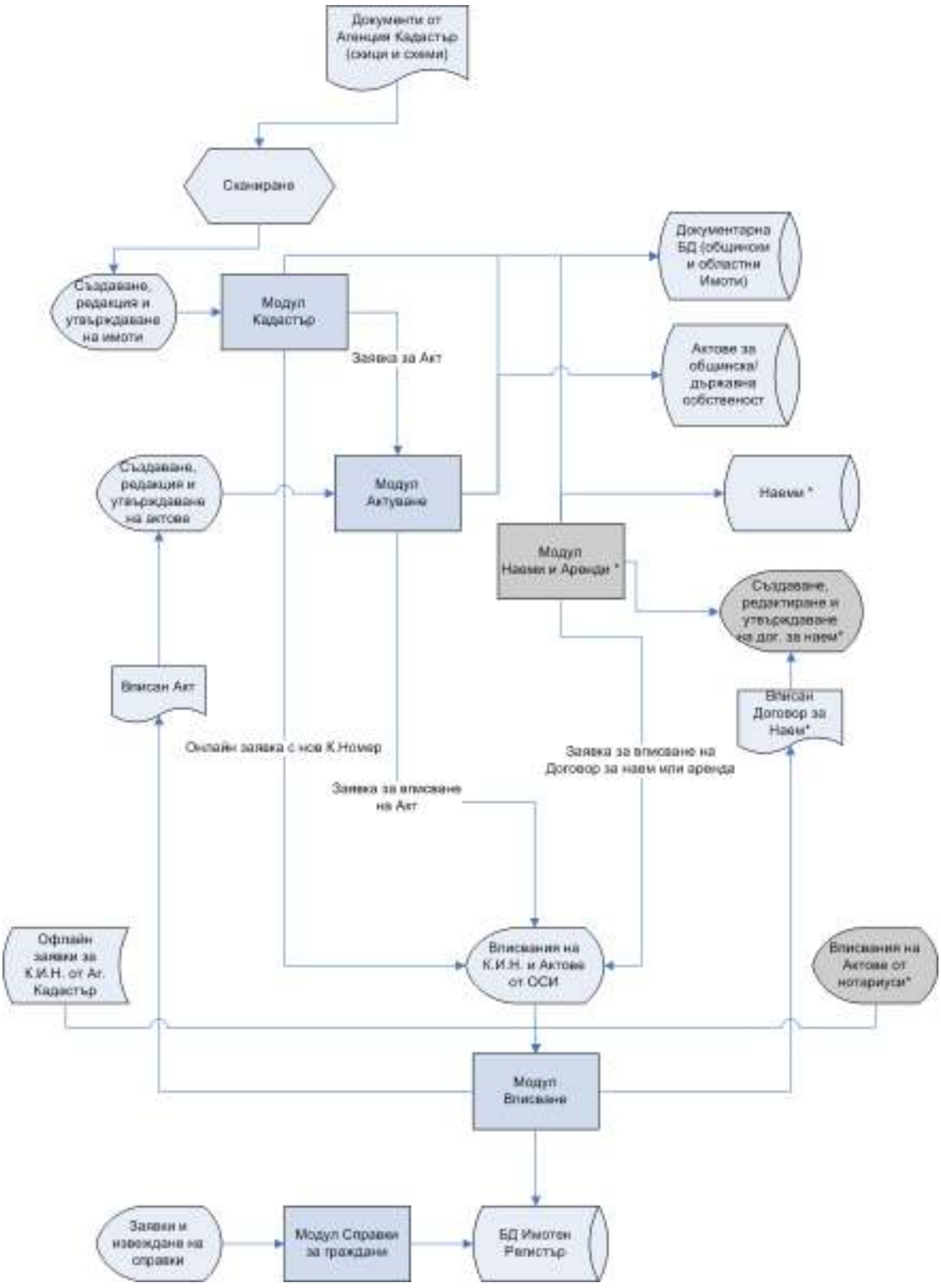
Комуникацията между агенцията по Кадастъра и ,,Имотния Регистър" е предвидено да става с офлайн файл, в който има пакет от заявки. Не се гарантира приемането на всички заявки за обработка. Заявките се преглеждат от чиновник по вписванията. Чиновникът или системата могат да отхвърлят някои заявки, като накрая трябва да се генерира файл с отхвърлените заявки.

Комуникацията между ОСИ и ИР е посредством обработката на онлайн заявки, които биват два вида:

- Онлайн заявка за нов(и) кадастрални номера (КИН)
- Онлайн заявка за вписване на акт

В даден момент от време може да се направи само една заявка.

Специализираните потребители могат да извършват всякакви справки в системите, но гражданите могат да правят справки само за кадастрална информация на имот. Специализираните потребители са всички потребители на системата без гражданите, като гражданите имат достъп само до ,,Имотния Регистър".



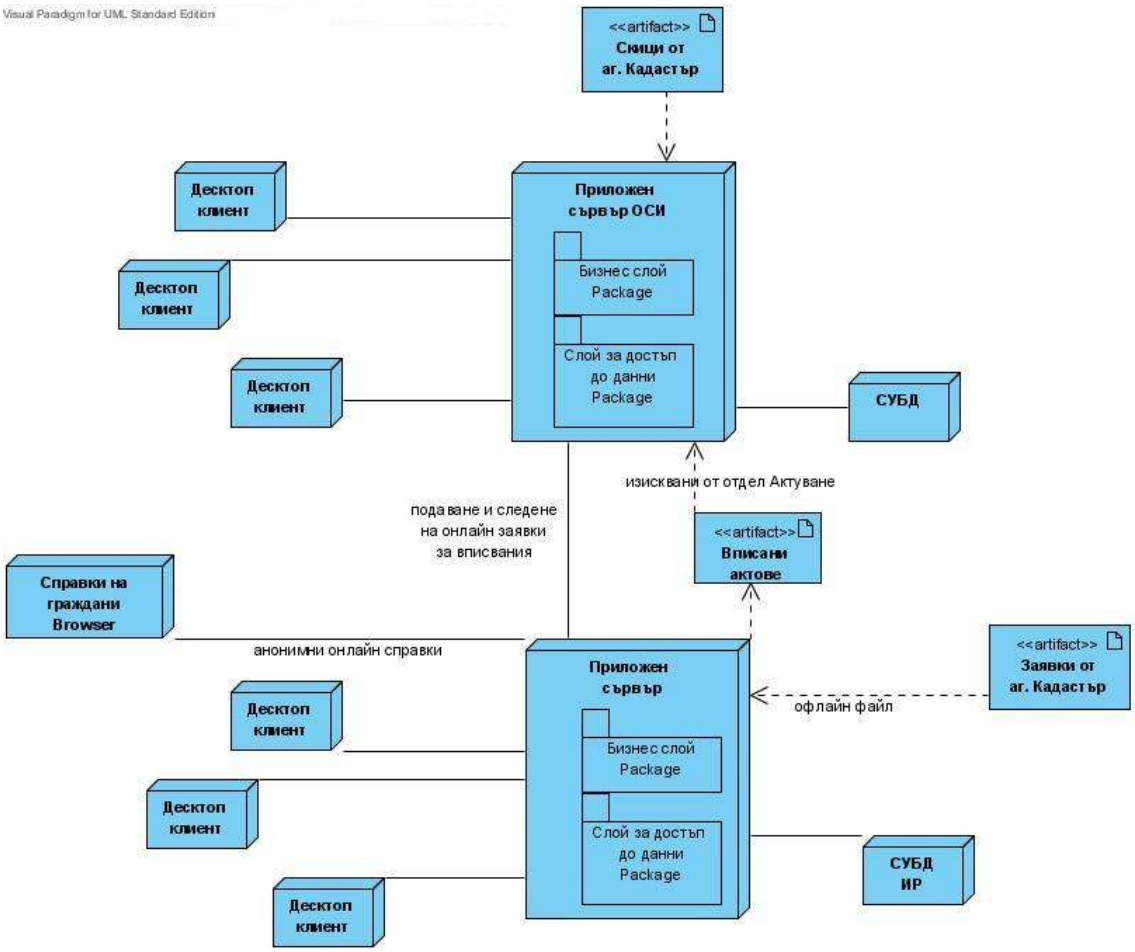
фиг. 4 Обща софтуерна архитектура на ОСИ и ИР

Изгледът на внедряването на системата е показан на фиг. 5. Той представя разположението на компонентите на отделните модули на системата по компютърни възли. UML диаграмата на внедряването следва общата архитектура, показана на фиг. 4. Двете системи - ОСИ и ИР, са представени като системи с трислойна софтуерна архитектура, като за всяка от системите клиентският слой (десктоп приложение) работи на една или повече клиентски станции, а сървърният слой - на приложни сървъри. Самият сървърен слой е изграден от два пакета - пакет Бизнес слой и пакет за Слой за достъп до данните.

Междусистемната комуникация се постига посредством два начина:

1. Подаване и следене на онлайн заявки - между ОСИ и ИР
2. Обработка на файл с офлайн заявки - от агенция Кадастър

Предвиден е и Уеб клиент за анонимни онлайн справки на граждани през Интернет браузър.

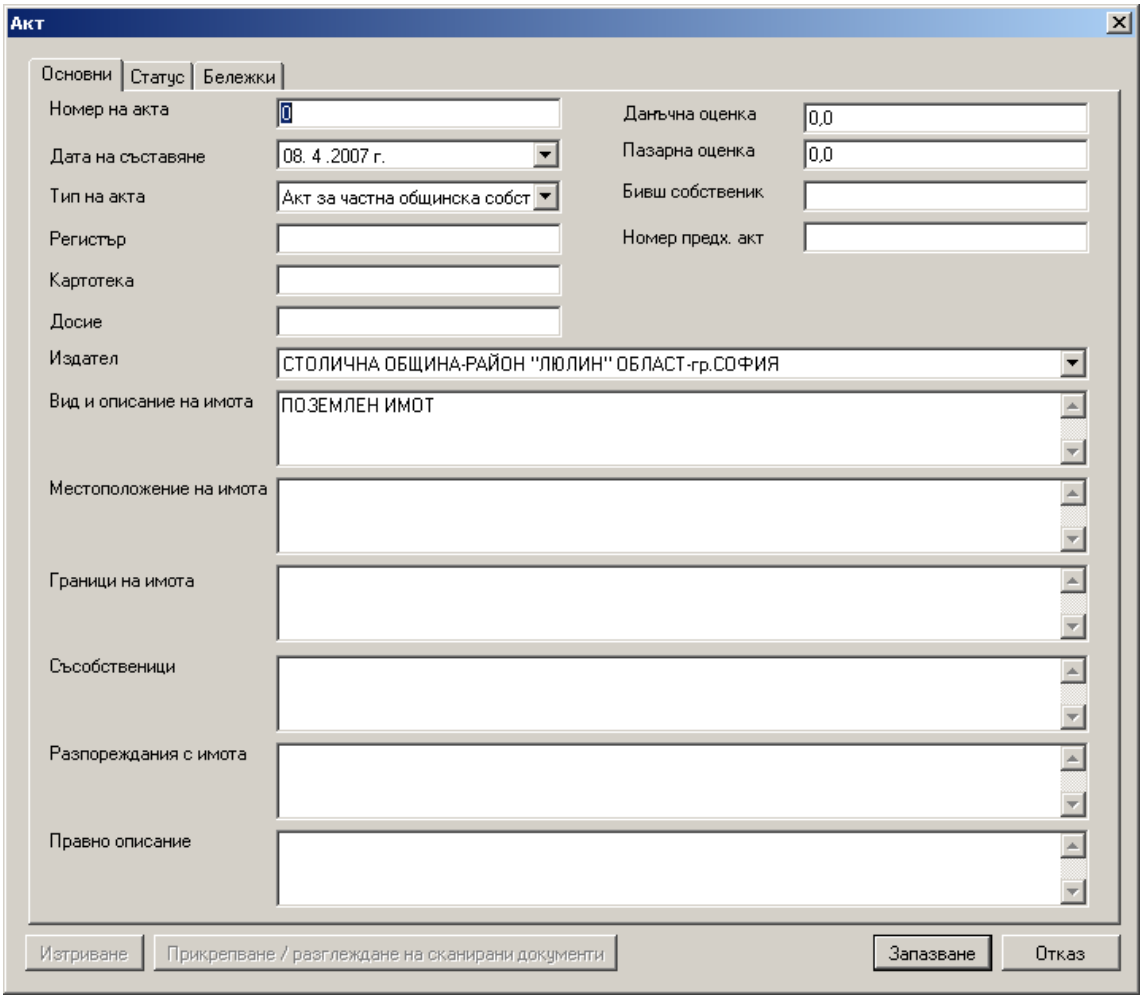


фиг. 5 Изглед на внедряването

4.3 Потребителски интерфейс

Потребителският интерфейс е изграден на базата на библиотеката Windows Forms, част от .NET framework. Интерфейса е проектиран като диалогово базирано Windows приложение, заради по-големите удобства които предлагат тези приложения. Друг фактор оказващ влияние е използването на приложението само в локалната мрежа на съответната държавна администрация.

Типичен диалогов прозорец от приложението е даден на следващата фиг. 6, която показва момент от редактирането на акт.



фиг. 6 Диалог за редактиране на акт

Всеки диалогов прозорец записва събраните от него данни в т. нар. entity обекти. Типичен пример за такъв entity обект е даден в Приложение 2, където са поместени извадки от изходния код на модула. Самото записване на данните от диалога в обектите се извършва чрез т. нар. механизъм на binding предоставен от Windows Forms. Следния код илюстрира свързването на отделните контроли от диалога за пропъртита на даден клас.

```
protected override void BindToObject(object entity)
{
    base.BindToObject(entity);

    BindTextBox(txtNumber, entity, "Number");
    BindTextBox(txtStatus, entity, "StatusName");
    BindDateTimePicker(dtpDateCreated, entity, "DateCreated");
    BindTextBox(txtLawReason, entity, "LawReason");
```

```
    BindTextBox(txtRegisterNumber, entity, "RegisterNumber");
    BindTextBox(txtCardIndexNumber, entity, "CardIndexNumber");
    BindTextBox(txtFileNumber, entity, "FileNumber");
    BindTextBox(txtDescription, entity, "Description");
    BindTextBox(txtLocation, entity, "Location");
    BindTextBox(txtBorders, entity, "Borders");
    BindTextBox(txtTaxAssesment, entity, "TaxAssesment");
    BindTextBox(txtMarketAssesment, entity, "MarketAssesment");
    BindTextBox(txtExOwners, entity, "ExOwners");
    BindTextBox(txtOwners, entity, "Owners");
    BindTextBox(txtExCertificates, entity, "ExCertificates");
    BindTextBox(txtOrders, entity, "Orders");
    BindTextBox(txtNotes, entity, "Notes");
    BindTextBox(txtRejectRemarks, entity, "RejectRemarks");
    BindTextBox(txtRegistrationNumber, entity, "RegistrationNumber");
    BindDateTimePicker(dtpRegistrationDate, entity, "RegistrationDate");
    BindTextBox(txtCreatedByUser, entity, "CreatedByUserFullname");
    BindTextBox(txtApprovedByUser, entity, "ApprovedByUserFullname");
    BindTextBox(txtSignedByUser, entity, "SignedByUserFullname");
    BindTextBox(txtLastUpdatedByUser, entity, "LastUpdatedByUserFullname");
    BindComboBox(cbType, "SelectedItem", entity, "DocumentType", "Name",
"ID",
        service.GetTypeList());
    BindComboBox(cbIssuer, "SelectedItem", entity, "Issuer", "Name", "ID",
        service.GetIssuerList());
}

protected void BindTextBox(TextBox textControl, object dataSource,
    string dataMember)
{
    textControl.DataBindings.Clear();
    textControl.DataBindings.Add("Text", dataSource, dataMember);
}

protected void BindComboBox(ComboBox combo, string propertyName,
    object bindingSource, string bindingMember, string displayMember,
    string valueMember, object dataSource)
{
    combo.DataSource = dataSource;
    combo.DisplayMember = displayMember;
    combo.ValueMember = valueMember;
    combo.DataBindings.Clear();
    combo.DataBindings.Add(propertyName, bindingSource, bindingMember,
false);
}

protected void BindDateTimePicker(DateTimePicker dateTimePickerControl,
    object dataSource, string dataMember)
{
    dateTimePickerControl.DataBindings.Clear();
    dateTimePickerControl.DataBindings.Add("Value", dataSource, dataMember);
}
```

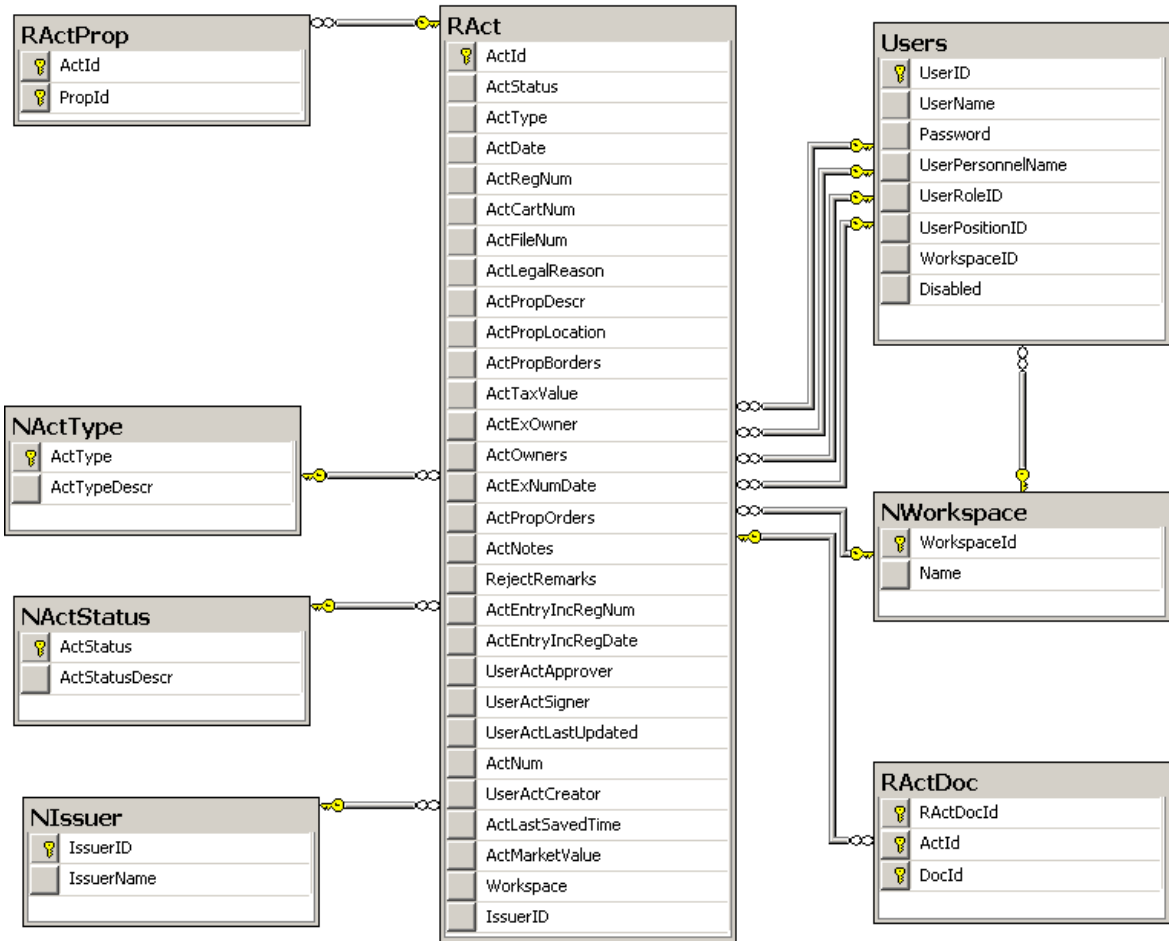
Повече информация за потребителския интерфейс при операционната система Windows може да бъде намерена в [5]

4.4 База от данни

Избраната базата от данни е Microsoft SQL Server 2005. Ключовите фактори, които надделяха при избора са:

- добрата интеграция с .NET framework
- безплатна версия, която при нужда може да се повиши до някоя от комерсиалните разгледани в обзора на Microsoft SQL Server
- Наличие на модерни средства за графично администриране
- Лесна поддръжка на базите данни

Част от модела на базата данни е специфициран от заданието предоставено от възложителя. Останалите таблици бяха създадени след анализ на изискванията на модула. При проектиране им бяха спазени правилата за нормализация на таблици. На следващата фиг. 7 е показан модела на релационната база от данни за модул „Актуване”.



фиг. 7 Диаграма на таблиците изграждащи модул актуване

4.4.1 Описание на таблиците

Таблица: RAct *описва даден акт*				
№	Поле	Тип	Описание	Забележка
1.	ActId	int	Идентификатор на акт (уникален)	primary key
2.	ActStatus	varchar(1)	Състояние на акт	foreign key към таблицата: **NActStatus**
3.	ActType	varchar(3)	Тип на акта	foreign key към таблицата: **NActType**
4.	ActDate	datetime	Дата на съставяне	
5.	ActRegNu	varchar(10)	Регистър	

	m			
6.	ActCartNum	varchar(10)	Картотека	
7.	ActFileNum	varchar(10)	Досие	
8.	ActLegalReason	text	Правно основание	
9.	ActPropDescr	text	Вид и описание на имота	
10.	ActPropLocation	text	Местоположение на имота	
11.	ActPropBorders	varchar(20)	Граници на имота	
12.	ActTaxValue	money	Данъчна оценка на имота	
13.	ActExOwner	text	Бивш собственик	
14.	ActOwners	text	Съсобственици	
15.	ActExNumDate	text	Номер и дата на предходен акт (актове)	
16.	ActPropOrders	text	Разпореждания с имота	
17.	ActNotes	text	Бележки	
18.	RejectRemarks	text	Забележки за отказ за утвърждаване	
19.	ActEntryIncRegNum	varchar(20)	Входящ номер на заявка за вписване	
20.	ActEntryIncRegDate	datetime	Дата на входящ номер на заявка за вписване	
21	UserActApprover	int	Идентификатор на потребител одобрил акта	foreign key към таблицата: **Users**

22	UserActSigner	int	Идентификатор на потребител подписал акта	foreign key към таблицата: **Users**
23	UserActLastUpdated	int	Идентификатор на потребител актуализирал последно акта	foreign key към таблицата: **Users**
24	ActNum	int	Номер на акта	
25	UserActCreator	Int	Идентификатор на потребител създал акта	foreign key към таблицата: **Users**
26	ActLastSavedTime	datetime	Дата и час на последна промяна на акта	
27	ActMarketValue	Money	Пазарна стойност на имота	
28	Workspace	int	Работно пространство на акта	foreign key към таблицата: **NWorkspace**
29	IssuerID	int	Идентификатор на издателя на акта	foreign key към таблицата: **NIssuer**

Таблица: NActType

номенклатура за тип на акт

№	Поле	Тип	Описание	Забележка
1.	ActType	varchar(3)	Код за тип на акта	primary key
2.	ActTypeDescr	varchar(50)	Описание на тип на акта	

Таблица: NIssuer

номенклатура за издател на акт

№	Поле	Тип	Описание	Забележка
1.	IssuerID	int	Код на издател	primary key
2.	IssuerName	varchar(50)	Име на издател	

Таблица: Users

потребители на системата				
№	Поле	Тип	Описание	Забележка
1.	UserID	int	Уникален идентификатор	primary key
2.	UserName	varchar(50)	Потребителско име	
3	Password	varchar(50)	Парола	
4	UserPersonnelName	varchar(50)	Име + Фамилия на потребитреля	
5	UserRoleID	int	Идентификатор на ролята	foreign key към таблицата: **UserRole**
6	UserPositionID	int	Идентификатор на заеманата позиция то потребителя	foreign key към таблицата: **UserPosition**
7	WorkspaceID	int	Идентифиактор на работното пространство	foreign key към таблицата: **NWorkspace**
8	Disabled	bit	Флаг за активност на потребителя	

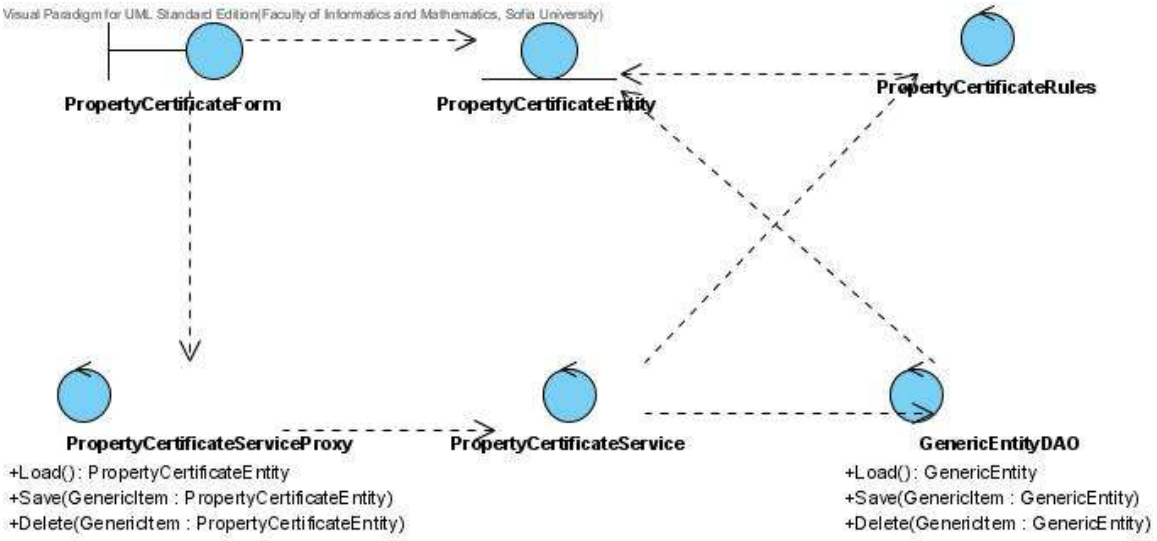
Таблица: NActStatus *номенклатура за състояние на акт*				
№	Поле	Тип	Описание	Забележка
1.	ActStatus	varchar(1)	Код за състояние на акт	primary key
2.	ActStatusDescr	varchar(50)	Описание на състояние на акт	

Таблица: RActProp *даден акт за кои имоти се отнася*				
№	Поле	Тип	Описание	Забележка
0.	PropId	int	Идентификатор на имот	foreign key към таблицата: **RProp**
1.	ActId	int	Идентификатор на акт	foreign key към таблицата: **RAct**

Повече информация за дизайна на бази от данни може да бъде намерена в [3, 11]

4.5 Главна клас диаграма

На следващата фиг. 8 е показана главната клас диаграма за модул актуване.



фиг. 8 Клас диаграма на модул „Актуване”

Предназначението на класовете е както следва:

- PropertyCertificateForm - Реализира потребителския диалог, чрез който се редактира даден акт. Представлява единична форма, изградена от няколко таба в които са групирани данните за акта. Реализацията се базира на Windows Forms.
- PropertyCertificateEntity - Съдържа всички атрибути на даден акт. Изгражда се чрез POCO (Plain Old C# Objects).
- PropertyCertificateServiceProxy – прокси клас, чрез който се достъпва уеб услугата за актуването.
- PropertyCertificateService - Предоставя интерфейс за отдалечено извикване на методи посредством Web услуга за модул актуване. В най-общия случай извиква Rules и DAO обектите - имплементация на шаблон Фасада.

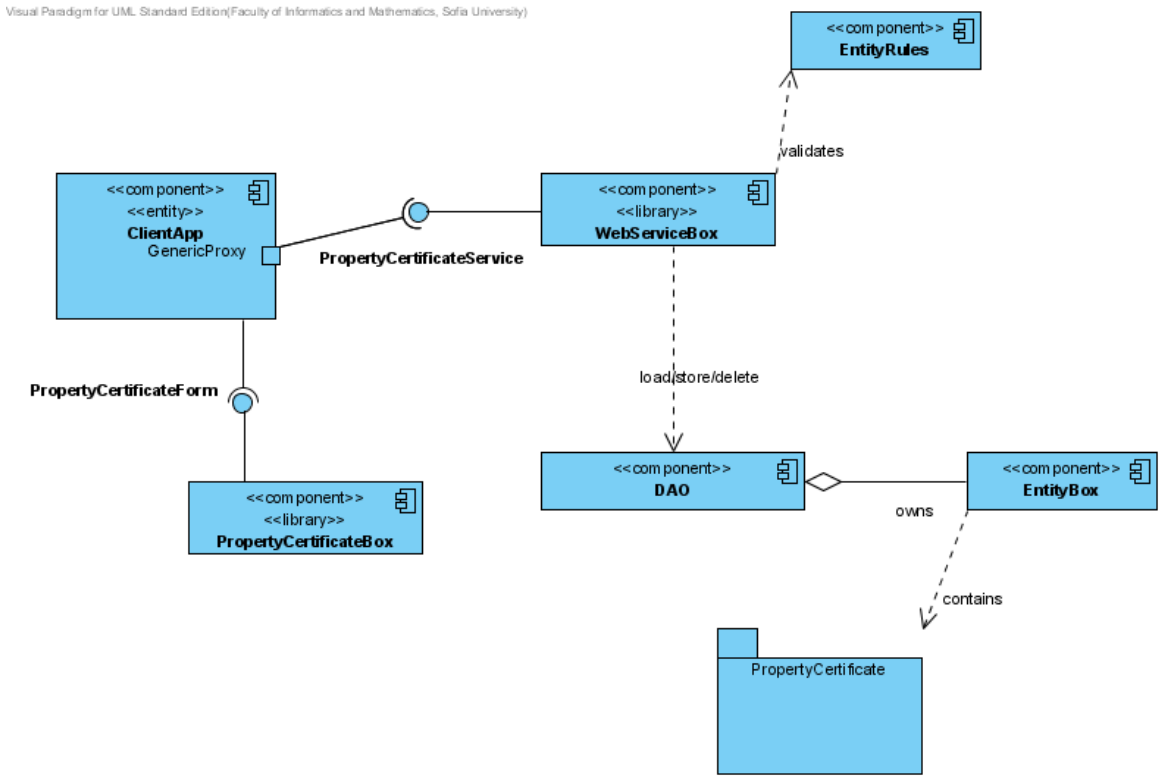
- PropertyCertificateRules - Съдържа всички ограничения, налагани върху даден акт. Пример: " Не може да бъде записан акт без всички необходими данни да бъдат въведени". Реализира се чрез C# клас.
- GenericEntityDAO - Шаблонен клас (generic), който предоставя интерфейс за работа с базата данни - CRUD функционалност.

Имплементацията на класовете е дадена в Приложение 2.

Повече информация за добрите практики при проектирането на класове може да бъде намерена в [6]

4.6 Компонентна диаграма

На следващата фиг. 9 е показана базовата компонентната диаграма за модула актуване.



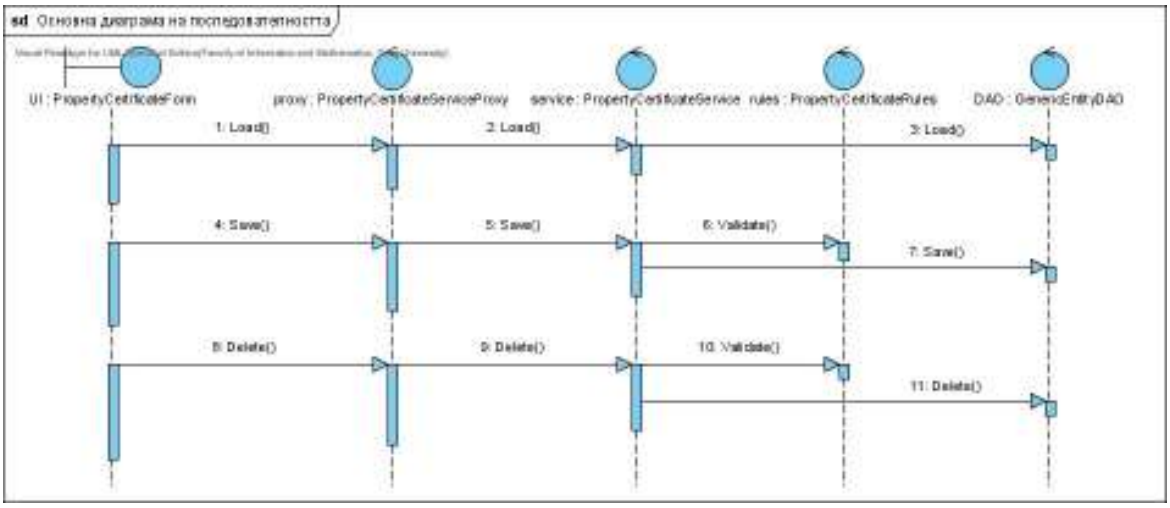
фиг. 9 Базова компонентна диаграма за модул Актуване

Предназначението на компонентите в горната диаграма е както следва:

- ClientApp – основен компонент на модула инсталиран при потребителя Съдържа главния диалог на приложението, менюто и т.н.
- PropertyCertificateBox – Съдържа диалоговите прозорци използвани за реализация на актуването.
- WebServiceBox – съдържа Web услугите, използвани за реализация на вариант актуване
- DAO –реализация шаблона Data Access Object, който е отговорен за манипулация на базата от данни
- EntityRules – състои се от класове, които отговарят за реализация на бизнес правилата заложени в системата
- EntityBox – съдържа класове, реализиращи атрибутите на акт.

4.7 Диаграма на последователността

На следващата фиг. 10 е показана примерна диаграма на последователността на извикванията между отделните класове в системата. Даден е пример за 3 случая: Зареждане на обект (Load), Записване (Save) и Изтриване (Delete) на обект.



фиг. 10 Диаграми на последователност

На диаграмата не е отбелязано, но трябва да се има предвид, че при всяко едно извикване на метод е възможно възникването на изключение (Exception), което е индикация за грешка.

4.8 Системни изисквания

4.8.1 Клиент

- **Операционна система** – Windows 98; Windows 98 SE, Windows 2000, Windows ME, Windows Server 2003, Windows XP, Windows Vista
- **Софтуер -** NET Framework 2.0 Redistributable, Microsoft Internet Explorer 6.0 със Service Pack 1 или по-нова версия, Microsoft Web Service Enhancements 3.0
- **Оперативна памет** – минимум 128 MB, препоръчително 256 MB или повече
- **Процесор** - тактова честота 400 MHz или повече
- **Дисково пространство** - минимум 800MB (включително 200MB за .NET Framework Redistributable)

4.8.2 Приложен сървър

- **Операционна система** – Windows 2000 Server; Windows Server 2003
- **Софтуер -** NET Framework 2.0 Redistributable, Microsoft Web Service Enhancements 3.0
- **Оперативна памет** – 1 GB или повече
- **Процесор** - тактова честота 1 GHz или повече
- **Дисково пространство** - минимум 2GB (включително 200MB за .NET Framework Redistributable)

4.8.3 Сървър за база от данни

- **Операционна система** – Windows 2000 Server; Windows Server 2003
- **Софтуер –** Microsoft SQL Server 2005
- **Оперативна памет** – 1 GB или повече
- **Процесор** - 1 GHz или повече
- **Дисково пространство** – 100GB или повече

5 Тестване и внедряване

Тестването на системата беше разделено на два етапа – тестване в изолирана среда и тестване в реални условия при клиента (пилотно тестване).

При първоначалния тест, който беше проведен в изолирана среда беше тествано основно всеки един от компонентите на системата, както и интеграцията между модулите. Тестовете бяха извършени с въвеждането на реални данни (реални актове). По време на самото тестване, бяха открити някой несъответствия с очакваното поведение, но заради добрата архитектура и следването на принципите на обектно-ориентираното програмиране, те бяха отстранени своевременно с минимални усилия.

Пилотно тестване на системата беше проведено на две места - общинско-областната администрация в гр. Враца и в Министерския Съвет на Република България. Системата е инсталирана на централен сървър в министерски съвет на следния адрес http://212.122.186.228/. Благодарение на поддръжката на отделни работни пространства (за области, общини и др.) се позволява работа както на служители на министерски съвет, така и на служители от община Враца. До момента служителите на двете ведомства са въвели няколко стотици имота и съответните актове към тях, с което е изпълнено пилотното тестване.

По време на пилотното тестване, потребителите дадоха изключително позитивни отзиви за работата на системата. Също така те предложиха някой изменения в проекта, които подобриха многократно процеса на работа в съответното ведомство. Пример за такава промяна е споменатата по-горе възможност за работа в отделни работни пространства.

Системата също така е представена и на Пловдивския технически панаир през 2007 година, където получи специална награда от Microsoft, състезавайки се с над 40 други проекта.

6 Заключение и насоки за бъдещо развитие

Дипломната работа разглежда проектирането и реализирането на модул за управление на процеса на актуване на държавни и общински имоти в Република България. Модулът е част от т.нар „Информационна система за управление на държавни и общински имоти (ИСУДОИ)”. Тази система представлява пакет от програмни приложения за управление на документи с данни за такива имоти, на регистри за кадастрална информация за тях, и на процеса по актуването им. ИСУДОИ предоставя интерфейс към система за имотен регистър, както и публичен достъп до издадените актове от страна на граждани, бизнес организации и държавната администрация на Република България.

Системата е базирана изцяло на Microsoft технологии. Изградена е с помощта на .NET Framework, като са използвани и библиотеки с отворен код като NHibernate за съхраняване на обекти в релационна база от данни. Избраният сървър за управление на релационни бази от данни е Microsoft SQL Server.

В момента се работи по пълното интегриране на разработената система със системата на имотния регистър. При интеграцията ще се проведе експериментална обработка на имотите и актовете, при което те ще бъдат вписани автоматично в системата на Имотния регистър, като се следва съществуващия процес на вписване. При интеграцията на двете системи ще се облекчи значително натоварването на служителите в държавната и общинска администрация.

Както беше показано и в принципната софтуерна архитектура, съществуват голям брой модули с които може да бъде разширена текущата система. Това са именно: Модул наеми и аренди, Хранилище за данни за наеми, Интерфейс за създаване и редактиране на договори

за наем, Интерфейс за вписвания на актове от нотариуси, Вписани договори за наем

7 Използвана литература

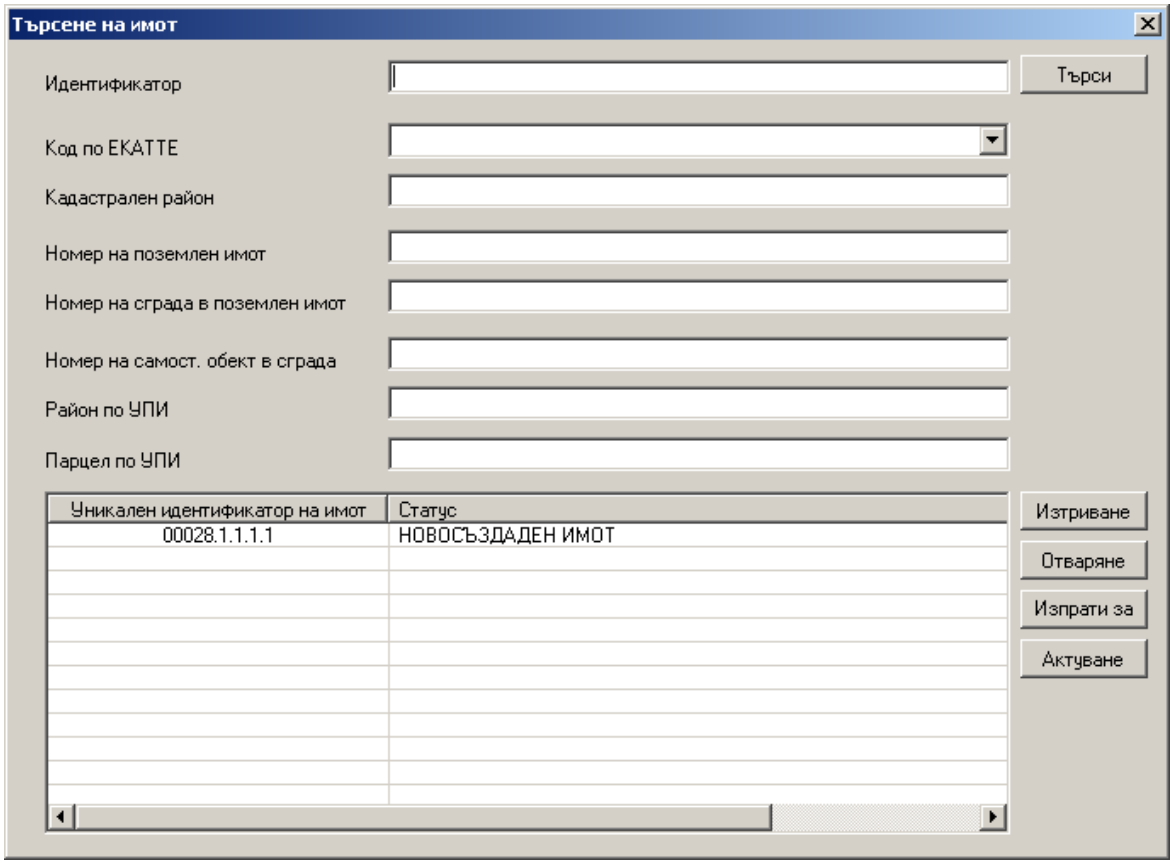
1. Джефри Рихтер, Microsoft .NET Framework – приложно програмиране, СофтПрес, 2002
2. Евлоги Георгиев, Научете сами SQL, Експрес дизайн, 1998
3. Майкъл Ернандес, Проектиране на бази от данни, СофтПрес, 2004
4. Мартин Фаулър, UML основи, СофтПрес, 2004
5. Everett McKay, Developing User Interfaces for Microsoft Windows, Microsoft Press, 1999
6. James W. Cooper, Introduction to Design Patterns in C#, IBM Watson Research Center, 2002
7. Jesse Liberty, Programming C#, 2nd edt., O'Reilly, 2002
8. Kalen Delaney, Inside SQL Server 2000, Microsoft Press, 2001
9. Microsoft Corporation, Microsoft SQL Serer Books Online
10. Microsoft Corporation, MSDN Library April 2006
11. Rebecca Riordan, Designing Relational Database Systems, Microsoft Press, 1999
12. Stephen Maguire, Writing Solid Code, Microsoft Press, 1993
13. Steve McConnel, Code Complete, Microsoft Press, 1993
14. Steve McConnel, Code Complete 2nd edt., Microsoft Press, 2004
15. Thuan Thai, Hoang Lam, .NET Framework Essentials 2nd edt., O'Reilly 2002

Приложение 1 – Ръководство на потребителя

Меню „Създаване на акт”

Търсене на имот

Преди създаването на даден акт, трябва да се намери имота, който ще се актува. Това се извършва чрез диалоговия прозорец показан на фиг. 11.



фиг. 11 Търсене на имот за актуване

Въвеждат се критериите за търсене в полетата „Идентификатор”, „Код по ЕКАТТЕ” и т.н. Достатъчно е да се въведе само едно поле. След това се натиска бутона „Търси” и намерените имоти се показват в таблицата.

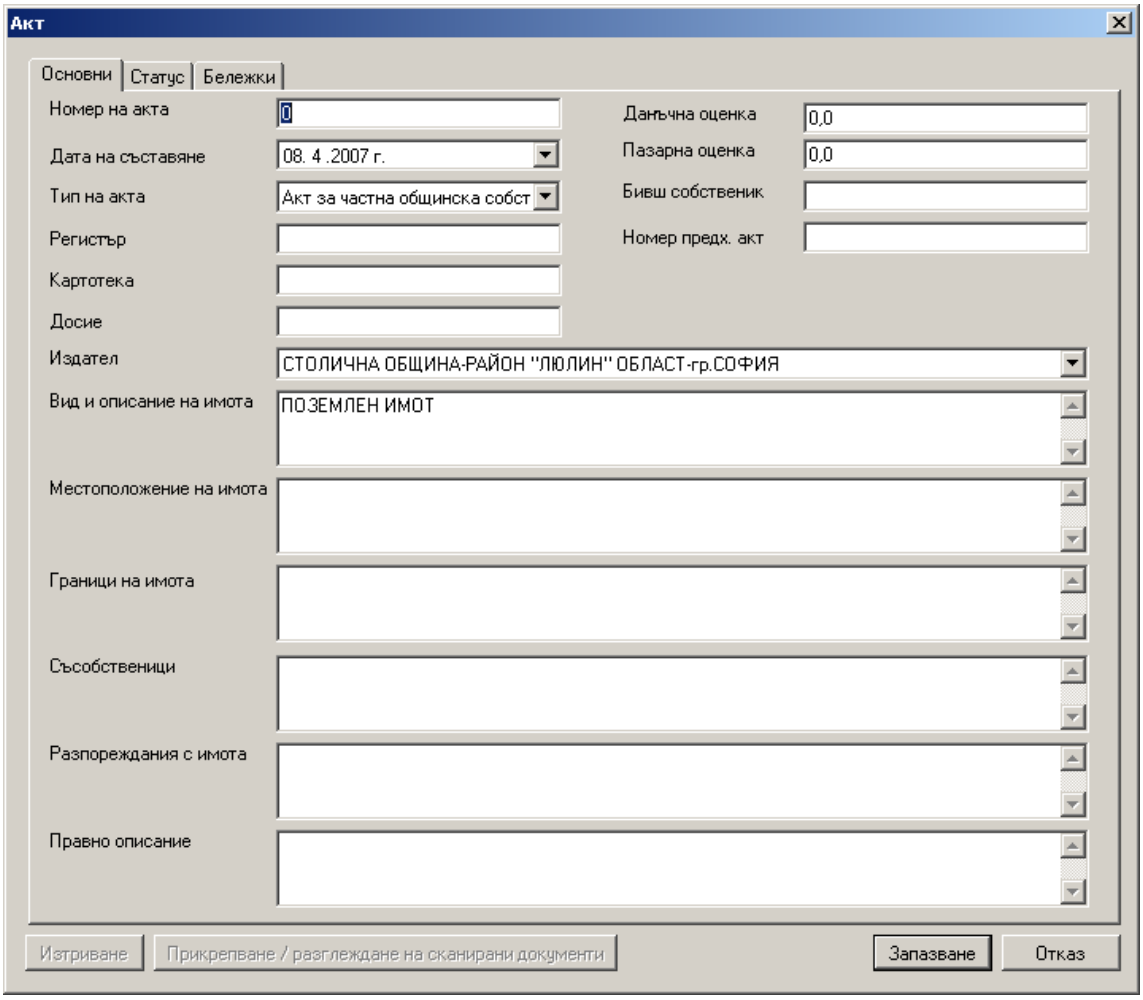
От така показаните имоти се избират един или няколко и се натиска бутона „Актуване”, с което се стартира процедурата по създаването на нов акт.

Бутон „Актуване”

Бутонът „Актуване” се използва за създаване на нови актове. Това става по следния начин:

- намират се имотите, за които ще се издава акт по начина описан в предходната точка – търсене на имот.
- избират се намерените имоти и се натиска бутона “Актуване”

В резултат на горните действия на екрана се появява следния диалогов прозорец за създаване на акт:



фиг. 12 Създаване на акт

Показаният диалогов прозорец на фиг. 12 е разделен на три части:

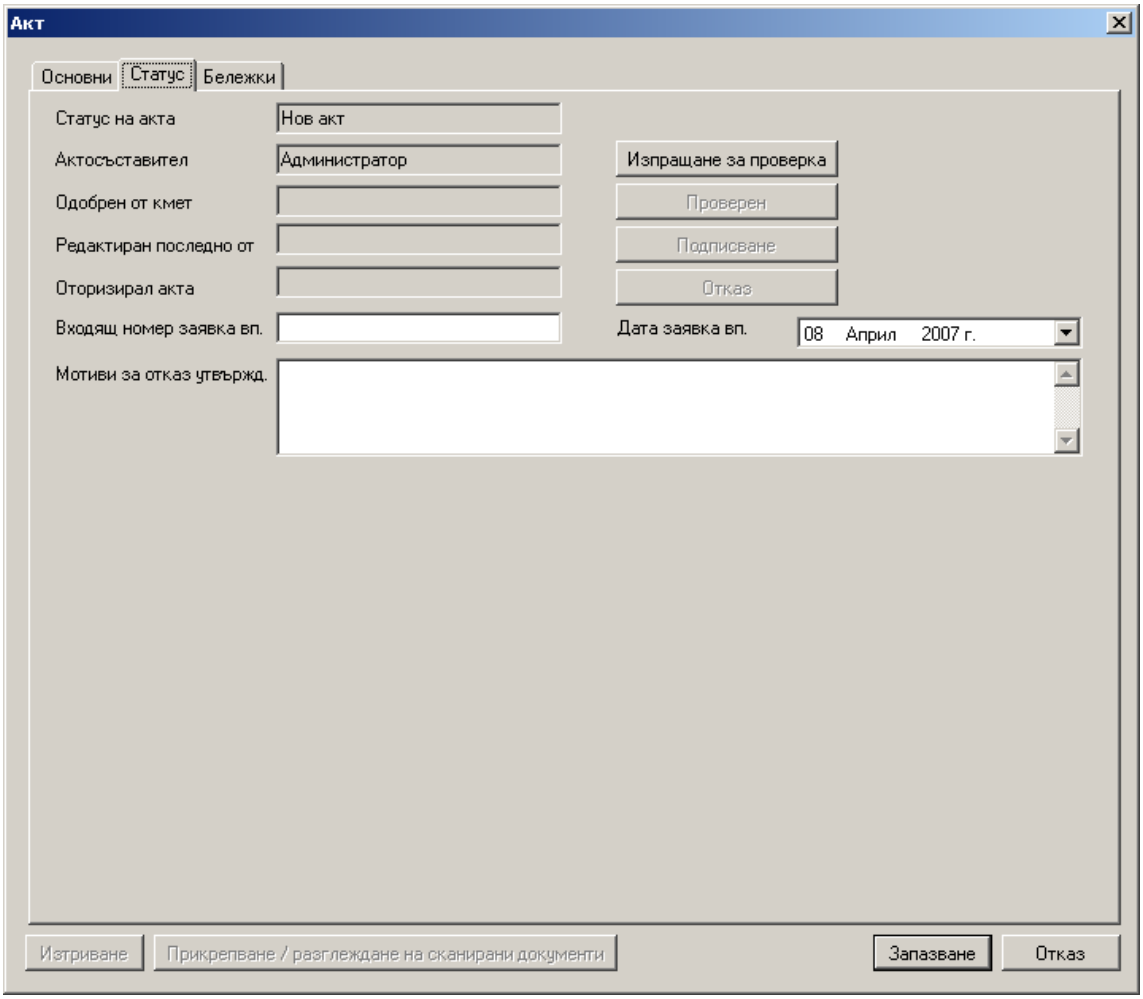
- основни характеристики – служи за въвеждане на основните данни за акта – номер, дата, граници и т.н.
- статус – съдържа данните за статуса на акта – одобрен, подписан и т.н.
- бележки – допълнителни бележки към акта

Раздел „Основни характеристики”

В настоящият раздел се въвеждат основните данни за акта. Полетата в които се въвеждат данни може да се разделят на три вида, според начина на попълването им:

- избор от падащ списък – такива са типа и издателя на акта
- въвеждане на числена стойност – данъчна и пазарна оценка. Системата не позволява въвеждане на данни символен тип данни.
- въвеждане на свободен текст – това са всички останали полета на акта (с изключение на избора на дата). За тях е разрешено въвеждането на всички видове символи.

Раздел „Статус”



фиг. 13 Статус на акт

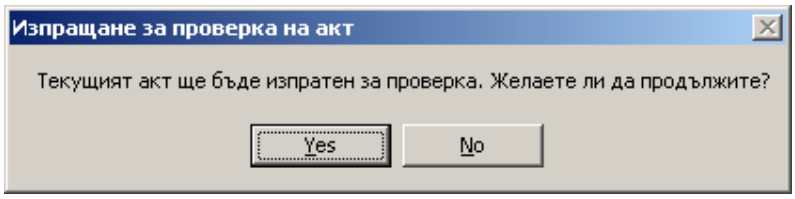
На фиг. 13 е показан диалога за статус на акта. В него потребителя може да преглежда или редактира състоянието в което се намира текущия акт в зависимост от правата, които притежава за работа с

актове. Полетата "Статус на акта", "Актосъставител", "Оторизирал акта", "Одобрен от кмет", "Редактиран последно от" се попълват автоматично от информационната система и потребителят не може да ги редактира. Единствените полета, които подлежат на редакция, ако са налични съответните права са: "Входящ номер на заявка за вписване", "Дата заявка за вписване" и "Мотиви за отказ утвърждаване".

Промяната на статуса на акта става чрез натискане на някой от бутоните "Изпращане за проверка", "Проверен", "Подписване", Отказ"". За да може да променя статуса на акта потребителя трябва да притежава съответните права. По-долу е разгледан по-подробно всеки един от случаите за промяна статуса на акта:

1. Изпращане на проверка

Първоначално, при самото си създаване акта се намира в състояние "Нов акт". След като служителят, създал акта приключи работата си по него, той натиска бутона "Изпращане за проверка", с което установява акта в състояние "За утвърждаване". Взети са мерки за предпазване от случайно натискане на бутона от потребителя, чрез извеждането на следното потвърждаващо съобщение, показано на фиг. 14:

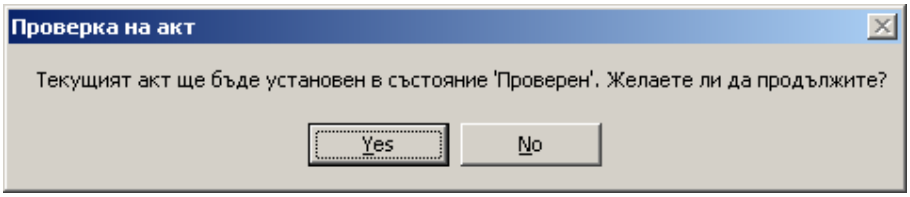


фиг. 14 Потвърждаващ диалог за изпращане на акт за проверка

Забележка: Горепоказаният диалог се отнася за операционна система Windows, която не е преведена на български език. Поради тази причина бутоните са именувани "Yes" и "No". При инсталиране на информационната система върху преведена на български език версия на операционната система, бутоните ще бъдат изведени с техните еквиваленти в българския език – "Да" и "Не".

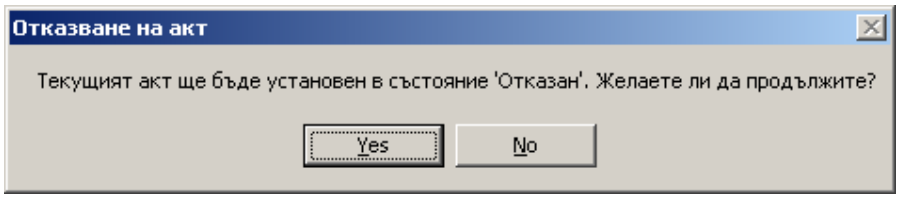
2. Задаване на статус на акта „Проверен"

След като един акт бъде установен в състояние "За утвърждаване", друг служител от администрацията проверява за коректността на въведените данни за акта. Ако всичко е въведено коректно, то тогава служителят използва бутона "Проверен" за да установи акта в състояние "Проверен". Отново се изисква от потребителя да потвърди действието си, чрез диалоговото съобщение, показано на фиг. 15.



фиг. 15 Потвърждаващ диалог за изпращане на акт за проверка

Ако при проверката си, служителя е открил някакви нередности по акта, то тогава той попълва мотивите си за отказ в полето "Мотиви за отказ утвърждаване" и го установява в състояние "Отказан", чрез натискане на бутона "Отказ". Отново се появява потвърждаващо диалогово съобщение, показано на фиг. 16

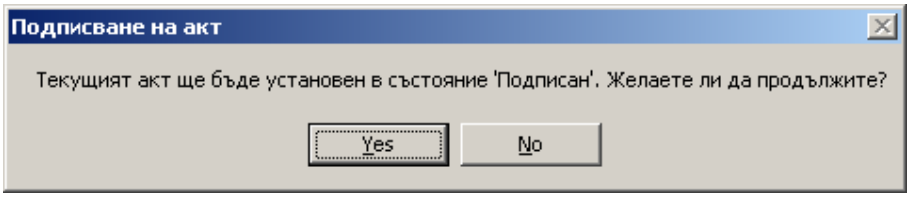


фиг. 16 Потвърждаващ диалог за установяване в състояние „Отказан"

При отказване на даден акт, той се връща за корекция на служителя, който първоначално го е създал.

3. Подписване

След като един акт бъде установен в състояние "Проверен", заемащият длъжността кмет го подписва. Това се отразява в системата чрез натискане на бутона "Подписване". Отново се изисква от потребителя да потвърди действието подписване, чрез показания на фиг. 17 диалогов прозорец.

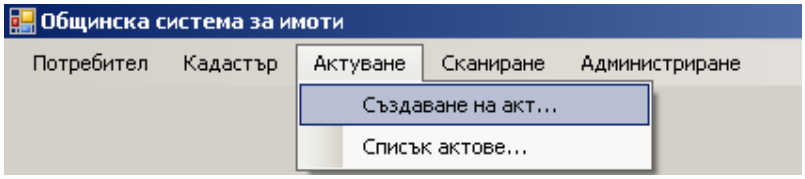


фиг. 17 Потвърждаващ диалог за установяване в състояние „Подписан”

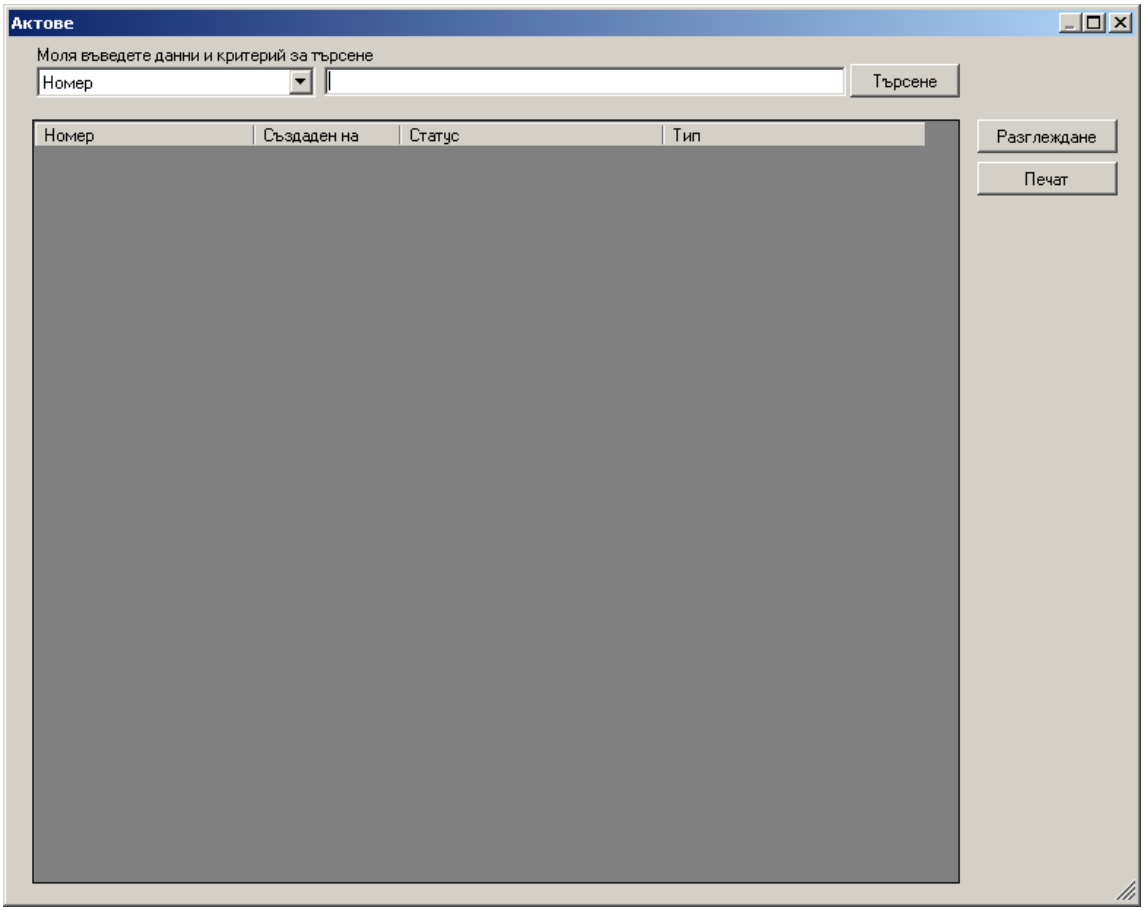
Ако поради някаква причина кмета не подпише дадения акт, то тогава чрез бутона “Отказ”, акта се установява в състояние “Подписването отказано”. При натискането на бутона “Отказ”, потребителя трябва да отговори утвърдително на показания диалогов прозорец, изискващ потвърждаване на отказа.

Меню „Списък актове”

След първоначалното въвеждане на акт в системата, последващата работа с акта (редактирането му и т.н.) става чрез менюто “Списък актове”, което е показано на фиг. 18. След неговия избор се появява диалоговия прозорец, показан на фиг. 19. В него потребителя може да търси актове, разглежда намерен акт и да отпечатва актове.



фиг. 18 Меню „Актуване”



фиг. 19 Списък актове

Търсене на акт

Търсенето на акт става по някой от следните 5 критерия: номер на акт, статус на акта, дата на създаване на акта, одобрил акта, подписал акта.

Избора на това по кой от всичките критерии ще се търси става от падащият списък в горния ляв ъгъл на диалоговия прозорец. След това в дясно от този списък се въвеждат данни за търсене (например номера на акт, ако се търси по номер на акт или името на подписалия акта, ако се търси по подписал акт).

След като се избере критерия за търсене и данните за него бъдат въведени се натиска бутона “Търсене”. Показалеца на мишката се превръща на пясъчен часовник и след кратко забавяне, списъка на намерените актове се появява в табличен вид. Потребителя може да

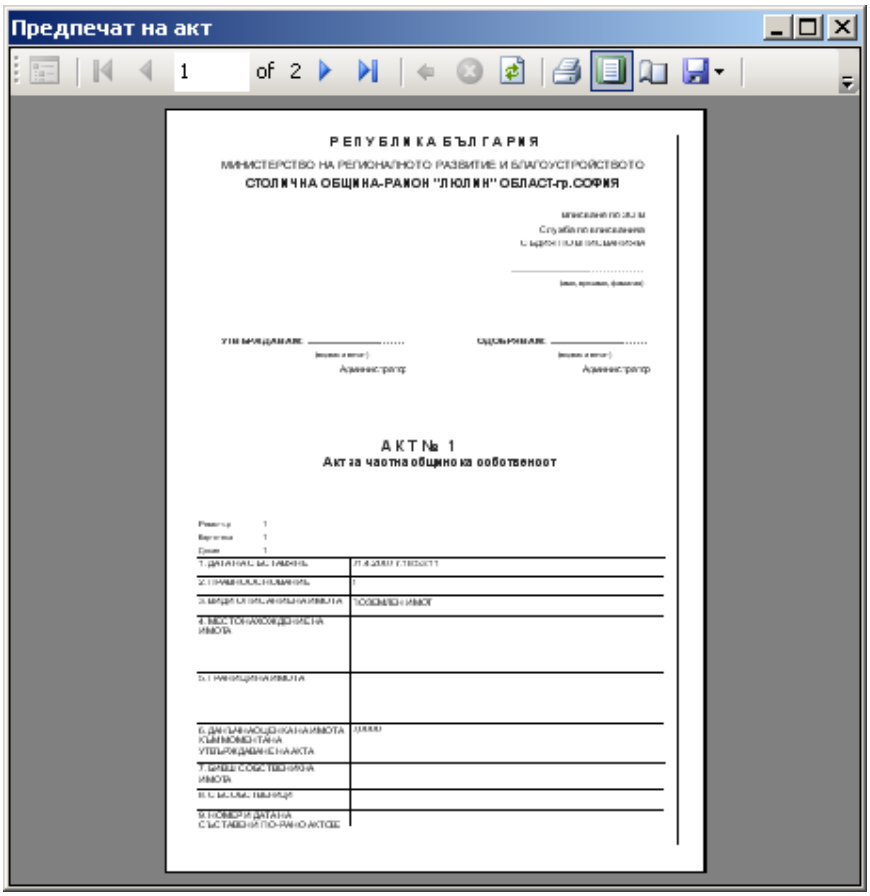
редактира или отпечата всеки от актовете в горния списък (ако има съответните права за това).

Разглеждане на акт

Разглеждането(редактирането) на акт става, чрез селектиране на акта от таблицата с намерените актове и натискане на бутона “Разглеждане”. В резултат на това се отваря диалога показан на фиг. 12, който вече беше описан в раздела за създаване на актове.

Печат на акт

Печата на акт става, чрез селектирането му от таблицата с намерени актове и натискане на бутона “Печат”. В резултат на това се отваря диалоговия прозорец за предпечат, който е показан на фиг. 20.



фиг. 20 Предпечат на акт

Диалогът за предпечат показва как би изглеждал акта ако се разпечата на хартиен носител. Самият печат става след като потребителя натисне върху иконата изобразяваща принтер от лентата с инструменти. За се извърши операцията успешно, към системата трябва да има инсталиран и конфигуриран правилно принтер.

Освен, че потребителя може да отпечата акта на хартиен носител, той може и да го съхрани във файл. Това става след като се натисне върху иконата изобразяваща дискета от лентата с инструменти.

Изтриване на акт

Информационната система позволява изтриване на актове, които са създадени погрешно. За целта потребителя трябва да има права за изтриване на актове. Актът също трябва да се намира в състояние, което позволява неговото изтриване – например "Нов акт".

За да бъде изтрит даден акт, то той първо трябва намерен в системата по метода описан в раздела "Търсене на акт". След това акта се отваря за разглеждане/редакция, както е описано в секцията "Разглеждане на акт".

След като се появи диалоговия прозорец за разглеждане/редактиране на акт, в долния ляв ъгъл се намира бутона за изтриване на текущия акт. Този бутон е разрешен само ако потребителя има съответните права за изтриване на акт и акта се намира в състояние, което позволява да бъде изтрит.

Приложение 2 – Изходен код на някой от по-важните класове

PropertyCertificate

Класът е типичен представител на т. нар. Entity класове. Той съдържа в себе си всички характеристики на даден акт, като уникален идентификатор, дата на издаване и т.н.

```
namespace Bonea.PropertyRegister.MunicipalityProperty.Entities
{
    public class PropertyCertificate
    {
        public PropertyCertificate()
        {
            mID = AppConst.UnknownID;
            mPropertyList = new List<Property>();
            mTaxAssesment = 0.0M;
            mMarketAssesment = 0.0M;
            mDateCreated = DateTime.Now;
            mRegistrationDate = DateTime.Now;
        }

        private int mID;
        private PropertyCertificateStatus mStatus;
        private PropertyCertificateType mDocumentType;
        private AppUser mCreatedByUser;
        private AppUser mApprovedByUser;
        private AppUser mSignedByUser;
        private AppUser mLastUpdatedByUser;
        private int mNumber;
        private DateTime mDateCreated;
        private string mRegisterNumber;
        private string mCardIndexNumber;
        private string mFileNumber;
        private string mLawReason;
        private string mDescription;
        private string mLocation;
        private string mBorders;
        private decimal mTaxAssesment;
        private decimal mMarketAssesment;
        private string mExOwners;
        private string mOwners;
        private string mExCertificates;
        private string mOrders;
        private string mNotes;
        private string mRejectRemarks;
        private string mRegistrationNumber;
        private DateTime mRegistrationDate;
        private DateTime mLastSavedTime;
        private PropertyCertificateIssuer mIssuer;
        private int mWorkspaceID;
        IList<Property> mPropertyList;
```

```
        public int ID
        {
            get { return mID; }
            set { mID = value; }
        }

        public PropertyCertificateStatus Status
        {
            get { return mStatus; }
            set { mStatus = value; }
        }

        public PropertyCertificateType DocumentType
        {
            get { return mDocumentType; }
            set { mDocumentType = value; }
        }

        public DateTime DateCreated
        {
            get { return mDateCreated; }
            set { mDateCreated = value; }
        }

        public string RegisterNumber
        {
            get { return mRegisterNumber; }
            set { mRegisterNumber = value; }
        }

        public string CardIndexNumber
        {
            get { return mCardIndexNumber; }
            set { mCardIndexNumber = value; }
        }

        public string FileNumber
        {
            get { return mFileNumber; }
            set { mFileNumber = value; }
        }

        public string LawReason
        {
            get { return mLawReason; }
            set { mLawReason = value; }
        }

        public string Description
        {
            get { return mDescription; }
            set { mDescription = value; }
        }

        public string Location
        {
            get { return mLocation; }
            set { mLocation = value; }
        }
```

```
        public string Borders
        {
            get { return mBorders; }
            set { mBorders = value; }
        }

        public decimal TaxAssesment
        {
            get { return mTaxAssesment; }
            set { mTaxAssesment = value; }
        }

        public decimal MarketAssesment
        {
            get { return mMarketAssesment; }
            set { mMarketAssesment = value; }
        }

        public string ExOwners
        {
            get { return mExOwners; }
            set { mExOwners = value; }
        }

        public string Owners
        {
            get { return mOwners; }
            set { mOwners = value; }
        }

        public string ExCertificates
        {
            get { return mExCertificates; }
            set { mExCertificates = value; }
        }

        public string Orders
        {
            get { return mOrders; }
            set { mOrders = value; }
        }

        public string Notes
        {
            get { return mNotes; }
            set { mNotes = value; }
        }

        public string RejectRemarks
        {
            get { return mRejectRemarks; }
            set { mRejectRemarks = value; }
        }

        public string RegistrationNumber
        {
            get { return mRegistrationNumber; }
            set { mRegistrationNumber = value; }
        }
```

```
        public DateTime RegistrationDate
        {
            get { return mRegistrationDate; }
            set { mRegistrationDate = value; }
        }

        public AppUser CreatedByUser
        {
            get { return mCreatedByUser; }
            set { mCreatedByUser = value; }
        }

        public AppUser ApprovedByUser
        {
            get { return mApprovedByUser; }
            set { mApprovedByUser = value; }
        }

        public AppUser SignedByUser
        {
            get { return mSignedByUser; }
            set { mSignedByUser = value; }
        }

        public AppUser LastUpdatedByUser
        {
            get { return mLastUpdatedByUser; }
            set { mLastUpdatedByUser = value; }
        }

        public int Number
        {
            get { return mNumber; }
            set { mNumber = value; }
        }

        public DateTime LastSavedTime
        {
            get { return mLastSavedTime; }
            set { mLastSavedTime = value; }
        }

        public PropertyCertificateIssuer Issuer
        {
            get { return mIssuer; }
            set { mIssuer = value; }
        }

        [XmlIgnore]
        public IList<Property> PropertyList
        {
            get { return mPropertyList; }
            set { mPropertyList = value; }
        }

        public Property[] SerializablePropertyList
        {
            get
            {
```

```
                return AppUtils.ListToFixedArray<Property>(mPropertyList);
            }
            set
            {
                mPropertyList = new List<Property>(value);
            }
        }

        public int WorkspaceID
        {
            get { return mWorkspaceID; }
            set { mWorkspaceID = value; }
        }

        public bool IsNew
        {
            get
            {
                if (ID == AppConst.UnknownID)
                {
                    return true;
                }
                else
                {
                    return false;
                }
            }
        }
    }
}
```

PropertyCertificate mapping

Даденият по-долу xml файл се използва от NHibernate за запис/четене на класа PropertyCertificate в/от релационната база данни.

```
<?xml version="1.0" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"

namespace="Bonea.PropertyRegister.MunicipalityProperty.Entities"

assembly="Bonea.PropertyRegister.MunicipalityProperty.Entities"
                   default-lazy="false">
  <class name="PropertyCertificate" table="RAct">
    <id name="ID" column="ActId" type="Int32" unsaved-value="-1">
      <generator class="identity" />
    </id>

    <timestamp name="LastSavedTime" column="ActLastSavedTime"/>
    <many-to-one name="Status" column="ActStatus"/>
    <many-to-one name="DocumentType" column="ActType"/>
    <many-to-one name="CreatedByUser" column="UserActCreator"/>
    <many-to-one name="ApprovedByUser" column="UserActApprover"/>
    <many-to-one name="SignedByUser" column="UserActSigner"/>
    <many-to-one name="LastUpdatedByUser" column="UserActLastUpdated"/>
    <many-to-one name="Issuer" column="IssuerID"/>
    <property name="Number" column="ActNum"/>
```

```
    <property name="DateCreated" column="ActDate"/>
    <property name="RegisterNumber" column="ActRegNum"/>
    <property name="CardIndexNumber" column="ActCartNum"/>
    <property name="FileNumber" column="ActFileNum"/>
    <property name="LawReason" column="ActLegalReason"/>
    <property name="Description" column="ActPropDescr"/>
    <property name="Location" column="ActPropLocation"/>
    <property name="Borders" column="ActPropBorders"/>
    <property name="TaxAssesment" column="ActTaxValue"/>
    <property name="MarketAssesment" column="ActMarketValue"/>
    <property name="ExOwners" column="ActExOwner"/>
    <property name="Owners" column="ActOwners"/>
    <property name="ExCertificates" column="ActExNumDate"/>
    <property name="Orders" column="ActPropOrders"/>
    <property name="Notes" column="ActNotes"/>
    <property name="RejectRemarks" column="RejectRemarks"/>
    <property name="RegistrationNumber" column="ActEntryIncRegNum"/>
    <property name="RegistrationDate" column="ActEntryIncRegDate"/>
    <property name="WorkspaceID" column="Workspace"/>

    <bag name="PropertyList" table="RActProp" cascade="none" lazy="true">
      <key column="ActId"/>
      <many-to-many column="PropId"
class="Bonea.PropertyRegister.MunicipalityProperty.Entities.Property,
Bonea.PropertyRegister.MunicipalityProperty.Entities" />
    </bag>
  </class>
</hibernate-mapping>
```

PropertyCertificateRules

Този клас съдържа всички ограничения (правила) които се налагат върху даден акт.

```
namespace Bonea.PropertyRegister.MunicipalityProperty.EntityRules
{
    public class PropertyCertificateRules
    {
        private PropertyCertificateRules()
        {
        }

        public static void ValidateSave(PropertyCertificate certificate)
        {
            List<RuleError> errors = new List<RuleError>();

            if (certificate.Number <= 0)
            {
                errors.Add(new RuleError("Number", "Невалиден номер на
акт"));
            }
            if (String.IsNullOrEmpty(certificate.RegisterNumber))
            {
                errors.Add(new RuleError("RegisterNumber", "Невалиден
регистър на акт"));
            }
            if (String.IsNullOrEmpty(certificate.FileNumber))
            {
```

```
                errors.Add(new RuleError("FileNumber", "Невалидно досие на
акт"));
            }
            if (String.IsNullOrEmpty(certificate.LawReason))
            {
                errors.Add(new RuleError("LawReason",
                    "Невалидно правно основание на акт"));
            }

            long certificateCount = GetPropertyCertificateCount(
                certificate.Number);
            if ((certificate.IsNew && (certificateCount > 0)) ||
                ((certificate.IsNew == false) && (certificateCount > 1)))
            {
                errors.Add(new RuleError("Number", "Вече съществува акт със
същия номер"));
            }

            if (errors.Count > 0)
            {
                throw new RuleException(errors);
            }
        }

        private static long GetPropertyCertificateCount(int number)
        {
            SessionManager dbSession = SessionManagerFactory.GetInstance();
            IQuery searchQuery = dbSession.Session.CreateQuery(
                "select count(pc) from PropertyCertificate as pc " +
                "where pc.Number = :Number");
            searchQuery.SetInt32("Number", number);

            long count = (long)searchQuery.UniqueResult();

            return count;
        }

        public static void ValidateDelete(PropertyCertificate certificate)
        {
            if (!((certificate.Status.StatusCode ==
PropertyCertificateStatusCodes.Created) ||
            (certificate.Status.StatusCode ==
PropertyCertificateStatusCodes.ApprovalRejected)))
            {
                throw new RuleException("NoProperty",
                     "Акта се намира в състояние в което неможе да бъде
изтрит");
            }
        }

        public static void ValidateChangeStatus(PropertyCertificate
certificate,
            PropertyCertificateStatusCodes newStatus)
        {
            switch (newStatus)
            {
                case PropertyCertificateStatusCodes.Created:
                    throw new RuleException("Status",
                         "Акта неможе да бъде установяван в състояние
'Създаден'");
```

```
                case PropertyCertificateStatusCodes.SendedForApproval:
                    if (!((certificate.Status.StatusCode ==
PropertyCertificateStatusCodes.Created) ||
                    (certificate.Status.StatusCode ==
PropertyCertificateStatusCodes.ApprovalRejected) ||
                    (certificate.Status.StatusCode ==
PropertyCertificateStatusCodes.SignRejected)))
                    {
                        throw new RuleException("Status",
                            "Акта неможе да бъде установяван в състояние 'За
утвърждаване' ако той не е 'Нов акт' 'Отхвърлен второ ниво' или 'Подписването
отказано'");
                    }
                    break;
                case PropertyCertificateStatusCodes.Approved:
                    if (certificate.Status.StatusCode !=
PropertyCertificateStatusCodes.SendedForApproval)
                    {
                        throw new RuleException("Status",
                            "Акта неможе да бъде установяван в състояние
'Утвърден второ ниво' ако той не е 'За утвърждаване'");
                    }
                    break;
                case PropertyCertificateStatusCodes.ApprovalRejected:
                    if (certificate.Status.StatusCode !=
PropertyCertificateStatusCodes.SendedForApproval)
                    {
                        throw new RuleException("Status",
                            "Акта неможе да бъде установяван в състояние
'Отхвърлен второ ниво' ако той не е 'За утвърждаване'");
                    }
                    break;
                case PropertyCertificateStatusCodes.Signed:
                    if (certificate.Status.StatusCode !=
PropertyCertificateStatusCodes.Approved)
                    {
                        throw new RuleException("Status",
                            "Акта неможе да бъде установяван в състояние
'Подписан' ако той не е 'Утвърден второ ниво'");
                    }
                    break;
                case PropertyCertificateStatusCodes.SignRejected:
                    if (certificate.Status.StatusCode !=
PropertyCertificateStatusCodes.Approved)
                    {
                        throw new RuleException("Status",
                            "Акта неможе да бъде установяван в състояние
'Подписването отказано' ако той не е 'Утвърден второ ниво'");
                    }
                    break;
                case PropertyCertificateStatusCodes.Registered:
                    if (certificate.Status.StatusCode !=
PropertyCertificateStatusCodes.Signed)
                    {
                        throw new RuleException("Status",
                            "Акта неможе да бъде установяван в състояние
'Вписан' ако той не е 'Подписан'");
                    }
                    break;
                case PropertyCertificateStatusCodes.RegistrationRejected:
```

```
                    if (certificate.Status.StatusCode !=
PropertyCertificateStatusCodes.Signed)
                    {
                        throw new RuleException("Status",
                            "Акта неможе да бъде установяван в състояние
'Отхвърлена заявка за вписване' ако той не е 'Подписан'");
                    }
                    break;
                default: throw new ArgumentException("Неизвестен статус на
акт");
            }
        }
    }
}
```

PropertyCertificateService

Показаният по-долу клас представлява уеб услугата, която реализира операциите извършвани с акт.

```
namespace Bonea.PropertyRegister.MunicipalityProperty.WebService
{
    /// <summary>
    /// Summary description for PropertyCertificate
    /// </summary>
    [WebService(Namespace =

"http://bonea.com/PropertyRegister/MunicipalityProperty/PropertyCertificate")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [ToolboxItem(false)]
    public class PropertyCertificateService : ServiceBase
    {
        private const String INVALID_WORKSPACE = "Нямате права за работа с
работното " +
            "пространство, към което принадлежи акта";

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateCreate, Authenticated =
true)]
        public PropertyCertificate New(int[] propertyIDList)
        {
            PropertyCertificate certificate = new PropertyCertificate();

            certificate.DateCreated = DateTime.Now;
            certificate.Status =
SessionContext.Session.Load<PropertyCertificateStatus>(
                AppConst.DefaultPropertyCertificateStatusID);
            certificate.DocumentType =
SessionContext.Session.Load<PropertyCertificateType>(
                AppConst.DefaultPropertyCertificateTypeID);
            certificate.Issuer = GetDefaultIssuer();

            AppUser emptyUser =
SessionContext.Session.Load<AppUser>(AppConst.EmptyUserID);
            certificate.CreatedByUser = CurrentUser;
            certificate.ApprovedByUser = emptyUser;
            certificate.SignedByUser = emptyUser;
```

```
            certificate.LastUpdatedByUser = emptyUser;
            certificate.WorkspaceID = CurrentUser.Workspace.ID;

            Property property;
            foreach (int propertyID in propertyIDList)
            {
                property = SessionContext.Session.Load<Property>(propertyID);
                certificate.PropertyList.Add(property);

                certificate.Description += property.PropType.PropTypeDescr
                    + Environment.NewLine;
                certificate.Borders += property.PropBorders +
Environment.NewLine;
                certificate.Location += BuildPropertyLocation(property)
                    + Environment.NewLine;
            }

            return certificate;
        }

        private PropertyCertificateIssuer GetDefaultIssuer()
        {
            IList<PropertyCertificateIssuer> issuerList = GetIssuerList();
            if (issuerList.Count > 0)
            {
                return issuerList[0];
            }
            return null;
        }

        private string BuildPropertyLocation(Property property)
        {
            StringBuilder location = new StringBuilder();
            AddPropertyLocationItem(location, property.PropLand, "местност:");
            AddPropertyLocationItem(location, property.PropPopulArea,
"маселено място:");
            AddPropertyLocationItem(location, property.PropMunicip,
"община:");
            AddPropertyLocationItem(location, property.PropRegion, "област:");
            AddPropertyLocationItem(location, property.PropDistr, "квартал:");
            AddPropertyLocationItem(location, property.PropBul, "булевард:");
            AddPropertyLocationItem(location, property.PropBulNum, "№");
            AddPropertyLocationItem(location, property.PropStr, "улица:");
            AddPropertyLocationItem(location, property.PropStrNum, "№");

            return location.ToString();
        }

        private const char WORD_SEPARATOR = ' ';

        private void AddPropertyLocationItem(StringBuilder location, string
item,
            string itemHeading)
        {
            if ((String.IsNullOrEmpty(item) == false) && (item.Trim() !=
String.Empty))
            {
                location.Append(itemHeading);
                location.Append(WORD_SEPARATOR);
                location.Append(item);
```

```
                location.Append(WORD_SEPARATOR);
            }
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateView, Authenticated =
true)]
        public PropertyCertificate Load(int ID)
        {
            PropertyCertificate entity =
SessionContext.Session.Load<PropertyCertificate>(ID);
            if (entity.WorkspaceID != CurrentUser.Workspace.ID)
            {
                throw new InvalidWorkspaceException(
                    INVALID_WORKSPACE);
            }

            return entity;
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateSave, Authenticated =
true)]
        public void Save(PropertyCertificate certificate)
        {
            PropertyCertificateRules.ValidateSave(certificate);

            if (certificate.WorkspaceID != CurrentUser.Workspace.ID)
            {
                throw new InvalidWorkspaceException(INVALID_WORKSPACE);
            }

            certificate.LastUpdatedByUser = CurrentUser;

            SessionContext.BeginTransaction(IsolationLevel.ReadCommitted);
            try
            {
                SessionContext.Session.SaveOrUpdate(certificate);

                SessionContext.CommitTransaction();
                SessionContext.Session.Flush();
            }
            catch
            {
                SessionContext.RollbackTransaction();
                SessionContext.Session.Clear();
                throw;
            }
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateDelete, Authenticated =
true)]
        public void DeleteByID(int ID)
        {
            PropertyCertificate entity =
SessionContext.Session.Load<PropertyCertificate>(ID);
```

```
            Delete(entity);
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateDelete, Authenticated =
true)]
        public void Delete(PropertyCertificate certificate)
        {
            PropertyCertificateRules.ValidateDelete(certificate);

            if (certificate.WorkspaceID != CurrentUser.Workspace.ID)
            {
                throw new InvalidWorkspaceException(INVALID_WORKSPACE);
            }

            LogPropertyCertificate logCertificate =
BuildLogPropertyCertificate(
                certificate);

            SessionContext.BeginTransaction(IsolationLevel.ReadCommitted);
            try
            {
                SessionContext.Session.Save(logCertificate);
                SessionContext.Session.Delete(certificate);

                SessionContext.CommitTransaction();
                SessionContext.Session.Flush();
            }
            catch
            {
                SessionContext.RollbackTransaction();
                SessionContext.Session.Clear();
                throw;
            }
        }

        private LogPropertyCertificate BuildLogPropertyCertificate(
            PropertyCertificate certificate)
        {
            LogPropertyCertificate logCertificate = new
LogPropertyCertificate();
            logCertificate.LogDate = DateTime.Now;
            logCertificate.LogActionType = "D";
            logCertificate.UserActualize = UserManager.GetAppUser(
                Thread.CurrentPrincipal.Identity.Name);
            logCertificate.ID = certificate.ID;
            logCertificate.Status = certificate.Status;
            logCertificate.DocumentType = certificate.DocumentType;
            logCertificate.DateCreated = certificate.DateCreated;
            logCertificate.RegisterNumber = certificate.RegisterNumber;
            logCertificate.CardIndexNumber = certificate.CardIndexNumber;
            logCertificate.FileNumber = certificate.FileNumber;
            logCertificate.LawReason = certificate.LawReason;
            logCertificate.Description = certificate.Description;
            logCertificate.Location = certificate.Location;
            logCertificate.Borders = certificate.Borders;
            logCertificate.TaxAssesment = certificate.TaxAssesment;
            logCertificate.MarketAssesment = certificate.MarketAssesment;
```

```
            logCertificate.ExOwners = certificate.ExOwners;
            logCertificate.Owners = certificate.Owners;
            logCertificate.ExCertificates = certificate.ExCertificates;
            logCertificate.Orders = certificate.Orders;
            logCertificate.Notes = certificate.Notes;
            logCertificate.RejectRemarks = certificate.RejectRemarks;
            logCertificate.RegistrationNumber =
certificate.RegistrationNumber;
            logCertificate.RegistrationDate = certificate.RegistrationDate;
            logCertificate.CreatedByUser = certificate.CreatedByUser;
            logCertificate.ApprovedByUser = certificate.ApprovedByUser;
            logCertificate.SignedByUser = certificate.SignedByUser;
            logCertificate.Number = certificate.Number;
            logCertificate.LastSavedTime = certificate.LastSavedTime;

            return logCertificate;
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateSave, Authenticated =
true)]
        public PropertyCertificate ChangeStatus(PropertyCertificate
certificate,
            PropertyCertificateStatusCodes newStatus)
        {
            PropertyCertificateRules.ValidateChangeStatus(certificate,
newStatus);
            if (certificate.WorkspaceID != CurrentUser.Workspace.ID)
            {
                throw new InvalidWorkspaceException(INVALID_WORKSPACE);
            }

            certificate.Status = GetStatus(newStatus);
            switch (newStatus)
            {
                case PropertyCertificateStatusCodes.Approved:
                    certificate.ApprovedByUser = CurrentUser;
                    break;
                case PropertyCertificateStatusCodes.Signed:
                    certificate.SignedByUser = CurrentUser;
                    break;
            }

            return certificate;
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateView, Authenticated =
true)]
        public PropertyCertificate[] GetList()
        {
            IQuery listQuery = SessionContext.Session.CreateQuery(
                "from PropertyCertificate as pc where pc.WorkspaceID =
:WorkspaceID");
            listQuery.SetInt32("WorkspaceID", CurrentUser.Workspace.ID);
            IList propertyCertificateList = listQuery.List();

            return
```

```
AppUtils.ListToFixedArray<PropertyCertificate>(propertyCertificateList);
        }

        [WebMethod]
        public PropertyCertificateStatus
GetStatus(PropertyCertificateStatusCodes status)
        {
            return
SessionContext.Session.Load<PropertyCertificateStatus>((int)status);
        }

        [WebMethod]
        public PropertyCertificateStatus[] GetStatusList()
        {
            IQuery statusQuery = SessionContext.Session.CreateQuery(
                "from PropertyCertificateStatus");
            IList statusList = statusQuery.List();

            return
AppUtils.ListToFixedArray<PropertyCertificateStatus>(statusList);
        }

        [WebMethod]
        public PropertyCertificateType[] GetTypeList()
        {
            IQuery typeQuery = SessionContext.Session.CreateQuery(
                "from PropertyCertificateType");
            IList typeList = typeQuery.List();

            return
AppUtils.ListToFixedArray<PropertyCertificateType>(typeList);
        }

        [WebMethod]
        public PropertyCertificateIssuer[] GetIssuerList()
        {
            IQuery typeQuery = SessionContext.Session.CreateQuery(
                "from PropertyCertificateIssuer");
            IList typeList = typeQuery.List();

            return
AppUtils.ListToFixedArray<PropertyCertificateIssuer>(typeList);
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateView, Authenticated =
true)]
        public PropertyCertificate[] SearchByNumber(int number)
        {
            IQuery searchQuery = SessionContext.Session.CreateQuery(
                "from PropertyCertificate as pc where (pc.Number = :Number)
AND (pc.WorkspaceID = :WorkspaceID)");
            searchQuery.SetInt32("Number", number);
            searchQuery.SetInt32("WorkspaceID", CurrentUser.Workspace.ID);

            IList resultsList = searchQuery.List();

            return
AppUtils.ListToFixedArray<PropertyCertificate>(resultsList);
```

```
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateView, Authenticated =
true)]
        public PropertyCertificate[] SearchByDateCreated(DateTime fromDate,
            DateTime toDate)
        {
            IQuery searchQuery = SessionContext.Session.CreateQuery(
                "from PropertyCertificate as pc " +
                "where ((pc.DateCreated >= :FromDate) AND (pc.DateCreated <=
:ToDate) " +
                "  AND (pc.WorkspaceID = :WorkspaceID))");
            searchQuery.SetDateTime("FromDate", AppUtils.TrimTime(fromDate));
            searchQuery.SetDateTime("ToDate", AppUtils.FillMaxTime(toDate));
            searchQuery.SetInt32("WorkspaceID", CurrentUser.Workspace.ID);

            IList resultsList = searchQuery.List();

            return
AppUtils.ListToFixedArray<PropertyCertificate>(resultsList);
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateView, Authenticated =
true)]
        public PropertyCertificate[] SearchByStatus(int statusID)
        {
            IQuery searchQuery = SessionContext.Session.CreateQuery(
                "from PropertyCertificate as pc " +
                "where (pc.Status.ID = :ID) AND (pc.WorkspaceID =
:WorkspaceID)");
            searchQuery.SetInt32("ID", statusID);
            searchQuery.SetInt32("WorkspaceID", CurrentUser.Workspace.ID);

            IList resultsList = searchQuery.List();

            return
AppUtils.ListToFixedArray<PropertyCertificate>(resultsList);
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateView, Authenticated =
true)]
        public PropertyCertificate[] SearchByApprovedUser(string fullname)
        {
            IQuery searchQuery = SessionContext.Session.CreateQuery(
                "from PropertyCertificate as pc " +
                "where (pc.ApprovedByUser.Fullname like :Name) " +
                "  AND (pc.WorkspaceID = :WorkspaceID)");
            searchQuery.SetString("Name", fullname + "%");
            searchQuery.SetInt32("WorkspaceID", CurrentUser.Workspace.ID);

            IList resultsList = searchQuery.List();

            return
AppUtils.ListToFixedArray<PropertyCertificate>(resultsList);
```

```
        }

        [WebMethod]
        [PrincipalPermission(SecurityAction.Demand,
            Role = UserCredentials.PropertyCertificateView, Authenticated =
true)]
        public PropertyCertificate[] SearchBySignedUser(string fullname)
        {
            IQuery searchQuery = SessionContext.Session.CreateQuery(
                "from PropertyCertificate as pc " +
                "where (pc.SignedByUser.Fullname like :Name) " +
                "  AND (pc.WorkspaceID = :WorkspaceID)");
            searchQuery.SetString("Name", fullname + "%");
            searchQuery.SetInt32("WorkspaceID", CurrentUser.Workspace.ID);

            IList resultsList = searchQuery.List();

            return
AppUtils.ListToFixedArray<PropertyCertificate>(resultsList);
        }
    }
}
```

SessionManager

Класът SessionManager се използва за управление на сесиите на NHibernate.

```
namespace Bonea.PropertyRegister.DataAccess
{
    public abstract class SessionManager
    {
        public SessionManager()
        {
        }

        static NHibernate.Cfg.Configuration mConfiguration = null;
        static volatile ISessionFactory mSessionFactory = null;
        static object mFactorySyncObject = new object();

        private const string HBM_ASSEMBLY_NAME_SECTION =
"HibernateMappingAssemblies";

        public abstract ISession Session
        {
            get;
        }

        public IDbConnection SessionConnection
        {
            get { return Session.Connection; }
        }

        public ISession NewSession()
        {
            ISessionFactory sessionFactory = GetSessionFactory();

            return sessionFactory.OpenSession();
```

```
        }

        public abstract void CloseSession();

        public abstract bool IsActiveSession
        {
            get;
        }

        private ISessionFactory GetSessionFactory()
        {
            if (mSessionFactory == null)
            {
                lock (mFactorySyncObject)
                {
                    if (mSessionFactory == null)
                    {
                        mConfiguration = new NHibernate.Cfg.Configuration();

                        List<string> mappingAssembliesList =
(List<string>)ConfigurationManager.GetSection(
                            HBM_ASSEMBLY_NAME_SECTION);
                        foreach (string assemblyName in
mappingAssembliesList)
                        {
                            mConfiguration.AddAssembly(assemblyName);
                        }

                        mSessionFactory =
mConfiguration.BuildSessionFactory();
                    }
                }
            }

            return mSessionFactory;
        }

        public void BeginTransaction(IsolationLevel isolation)
        {
            Session.BeginTransaction(isolation);
        }

        public void CommitTransaction()
        {
            Session.Transaction.Commit();
        }

        public void RollbackTransaction()
        {
            Session.Transaction.Rollback();
        }

        public bool IsActiveTransaction
        {
            get
            {
                return ((Session.Transaction != null) &&
(Session.Transaction.IsActive));
            }
        }
```

```
    }
}

namespace Bonea.PropertyRegister.DataAccess
{
    public class PerRequestSessionManager : SessionManager, IHttpModule
    {
        internal PerRequestSessionManager()
        {
        }

        private const string NHIBERNATE_SESSION_KEY = "NHIBERNATE_SESSION";

        public override ISession Session
        {
            get
            {
                ISession resultSession;

                if
(HttpContext.Current.Items.Contains(NHIBERNATE_SESSION_KEY))
                {
                    resultSession =

(ISession)HttpContext.Current.Items[NHIBERNATE_SESSION_KEY];
                }
                else
                {
                    resultSession = NewSession();
                    HttpContext.Current.Items[NHIBERNATE_SESSION_KEY] =
resultSession;
                }

                return resultSession;
            }
        }

        public override bool IsActiveSession
        {
            get { return
HttpContext.Current.Items.Contains(NHIBERNATE_SESSION_KEY); }
        }

        public override void CloseSession()
        {
            if (IsActiveSession == true)
            {
                if (IsActiveTransaction == true)
                {
                    RollbackTransaction();
                }

                Session.Close();
                HttpContext.Current.Items.Remove(NHIBERNATE_SESSION_KEY);
            }
        }

        #region IHttpModule

        public void Init(HttpApplication context)
```

```
        {
            context.EndRequest += new EventHandler(OnEndRequest);
        }

        public void Dispose()
        {
            //Intentionaly not implemented
        }

        public void OnEndRequest(Object sender, EventArgs e)
        {
            CloseSession();
        }

        #endregion
    }
}
```

SqlServerTokenManager

Показаният по-долу клас се използва за автентикация на даден потребител преди извикване на метод от уеб услуга. При успешно разпознаване на предоставяните от потребителя белези за автентикация, то правата, които има за работа със системата се асоциират с текущата нишка.

```
namespace Bonea.PropertyRegister.WebServiceCommon
{
    [SecurityPermissionAttribute(SecurityAction.Demand,
        Flags = SecurityPermissionFlag.UnmanagedCode)]
    public class SqlServerTokenManager : UsernameTokenManager
    {
        public SqlServerTokenManager()
        {
        }

        public SqlServerTokenManager(XmlNodeList nodes)
            : base(nodes)
        {
        }

        protected override string AuthenticateToken(UsernameToken token)
        {
            if (UserManager.LogIn(token.Username, token.Password) == false)
            {
                throw new ApplicationException("Not a valid user");
            }

            GenericIdentity identity = new GenericIdentity(token.Username);
            GenericPrincipal principal = new GenericPrincipal(identity,
                UserManager.GetUserCredentials(token.Username));

            Thread.CurrentPrincipal = principal;

            return token.Password;
        }
```

```
        }
}
```

Приложение 3 - Подготовка на работно място

Цел

Настоящото ръководство е създадено с цел уеднаквяване и по-лесно подготвяне на работно място за проекта „Информационната система за управление на държавни и общински имоти (ИСУДОИ)". В него се съдържат инструкции за мястото за сваляне, версиите и спецификите при инсталирането на програмните продукти необходими за разработване на системата.

Продукти и тяхната инсталация

1. Инсталирайте **Microsoft Visual Studio 2005 Professional** или по-пълна версия + помощната система към него - **MSDN**
2. Инсталирайте **Service Pack 1 for Visual Studio 2005**, като предварително го свалите от http://msdn.microsoft.com/vstudio/support/vs2005sp1/default.aspx
3. Инсталирайте **Microsoft SQL Server 2005 Express Edition**, като предварително го свалите от http://www.microsoft.com/downloads/details.aspx?FamilyID=220549b5-0b07-4448-8848-dcc397514b41&DisplayLang=en
4. Инсталирайте **Microsoft SQL Server Management Studio Express**, като предварително го свалите от http://www.microsoft.com/downloads/details.aspx?FamilyID=c243a5ae-4bd1-4e3d-94b8-5a0f62bf7796&DisplayLang=en
5. Инсталирайте по желание **Microsoft SQL Server 2005 Express Edition Service Pack 1**, като предварително го свалите от http://www.microsoft.com/downloads/details.aspx?FamilyID=11350b1f-8f44-4db6-b542-4a4b869c2ff1&DisplayLang=en
6. Инсталирайте библиотеката **Microsoft Web Services Enhancements (WSE) 3.0 for Microsoft .NET**, като

предварително я свалите от http://www.microsoft.com/downloads/details.aspx?familyid=018A09FD-3A74-43C5-8EC1-8D789091255D&displaylang=en

7. Инсталирайте библиотеката **NHibernate 1.2** (beta), като предварително я свалите от http://www.hibernate.org/343.html

Специфични настройки за потребител

1. Отворете файла *MunicipalityProperty-> WebService-> Web.config* и задайте името на SQL Server-а и името на базата данни, като промените "**data source**" и "**initial catalog**" от ключа „**hibernate.connection.connection_string**" намиращ се в секцията "**nhibernate**"
2. Настройте проектите, които да се стартират:
   a. От контекстното меню за solution файла в *Solution Explorer*-а на Visual Studio изберете "**Set StartUp Projects...**"
   b. В дървото разположено в ляво на екрана изберете „**Startup project**"
   c. Изберете от в дясната част на диалога "**Multiple Startup Projects**"
   d. Задайте "**Action**"-а за "**Web Service**" и "**Windows Client**" да бъде "**Start**"

Приложение 4 - Конвенция за кода

С цел уеднаквяване на стила на кода се въвежда конвенция, която включва редица препоръки за форматирането, имената на типове, членове и променливи, елементи от потребителския интерфейс и други.

Повече информация за добрите практики при писането на код може да се намери в [12, 13, 14]

Константите се пишат с главни букви

```
private const int MAX_VALUE = 4096;
private const string INPUT_FILE_NAME = "input.xml";
```

Член-променливите се пишат с префикс "m"

```
private Hashtable mUsersProfiles;
private ArrayList mUsers;
```

Именуване на идентификатори

Идентификатор	Стил	Пример
пространство от имена (namespace)	Pascal Case	**System.Windows.Forms**
тип (клас, структура, ...)	Pascal Case	**TextWriter**
интерфейс (interface)	Pascal Case, префикс "**I**"	**ISerializable**
изброен тип (enum type)	Pascal Case	**FormBorderStyle**
изброена стойност (enum value)	Pascal Case	**FixedSingle**
поле само за четене (read-only field)	Pascal Case	**UserIcons**
поле-константа (constant)	UPPERCASE	**MAX_VALUE**
свойство (property)	Pascal Case	**BorderColor**
събитие (event)	Pascal Case	**SizeChanged**
метод (method)	Pascal Case	**ToString()**
член-променлива (field)	Pascal Case, префикс "**m**"	**mUserProfiles**
статична член-променлива (static field)	Pascal Case, префикс "**m**"	**mTotalUsersCount**
параметър на метод (parameter)	Pascal Case	**fileName**
локална променлива (local variable)	Camel Case	**currentIndex**

Именуване на контроли

WinForms Контрол	Префикс
Button	btn
CheckBox	chk
CheckedListBox	cbl
ColorDialog	dlg
ComboBox	cb
ContextMenuStrip	mnu
DataGridView	gv
DateTimePicker	dtp
ErrorProvider	err
FontDialog	dlg
GroupBox	gb
ImageList	iml
Label	lbl
ListBox	lb
ListView	lv
OpenFileDialog	dlg
MaskedTextBox	txt
MenuStrip	mnu
MonthCalendar	cal
NumericUpDown	nud
PictureBox	img
PrintDialog	dlg
ProgressBar	pb
Panel	pnl
ProgressBar	prb
RadioButton	rb
RichTextBox	txt
StatusStrip	sb
SaveFileDialog	dlg
ToolStrip	tb
TabControl	tab
ToolTip	tt
Timer	tmr
TreeView	tv
TextBox	txt

Web Контрол	Префикс
Button	btn
Calendar	cal
CheckBox	chk
CheckedBoxList	cbl
GridView	gv
DataList	dl
DetailsView	dv
DropDownList	cb
FormView	fv
HyperLink	lnk
Image	img
ImageButton	btn
ImageMap	imm
Label	lbl
LinkButton	btn
ListBox	lb
ObjectDataSource	ods
RadioButton	rb
RadioButtonList	rbl
RangeValidator	vld
RegularExpressionValidator	vld
Repeater	rpt
RequiredFieldValidator	vld
SiteMapDataSource	sds
TextBox	txt
ValidationSummary	vs
XmlDataSource	xds