



Софийски университет “Св. Климент Охридски”
Факултет по Математика и Информатика
Катедра Информационни Технологии



Дипломна работа

Тема:

**Изграждане на информационна система за координиране,
регистриране и отчитане на дейността на техници във
„Фонте Фреско” ООД**

Дипломант:

Биляна Милинова Бамбалова

Специалност: Софтуерни технологии

Фак. № М21949

Научен ръководител:

доц. д-р Боян Бончев

София, 2007

Съдържание

I Увод	4
1.Цел на дипломната работа	4
2.Структура на дипломната работа	4
I Описание на предметната област и потребителските изисквания	6
II Използвани технологии	10
1. .NET Framework.....	10
1.1. Common Language Runtime	11
1.2. Framework Class Library	12
2. .NET Compact Framework	15
2.1. Предимства и цели	16
2.2. SQL CE	16
2.3. Разлики с .NET Framework.....	17
2.4. Стратегии за достъп до данни.....	19
3. Web услуги.....	21
III Трислойна архитектура на приложението	23
1.Трислоен модел на комуникация.....	23
1.1. Функции на слоевете.	23
1.2. Предимства на трислойната архитектура	24
2.Трислойна архитектура на приложението.....	25
2.1. Представителен слой.....	25
2.2. Бизнес слой	26
2.3. Слой на базата данни.	26
3. Модел на имплементация на системата (Implementational model)	27
IV Проектиране и реализация на клиентския слой	28
1. Модел на случаи на употреба- (Use Case) модел	28
1.1. Модел на случаите на употреба за FonteFrescoMobile приложението	28
1.2. Модел на случаите на употреба за FonteFrescoService приложението	34
2. Имплементация на FonteFrescoMobile	38
2.1. Потребителски интерфейс	38
2.2. Реализация на offline режима.....	44
2.3. Реализация на online режима	48
2.4. User контроли	54
2.5. Utils класове	56
2.6. Динамична реализация.....	57
2.7. Структура на приложението	60
3. Имплементация на FonteFrescoService	61
3.1. Потребителски интерфейс	61
3.2. Извикване на Web услуга.....	69
3.3. Структура на приложението	71
V Реализация на базата данни	73
1. Генериране на кода на базата данни	73
2. Логическа структура.....	73
VI Внедряване	87
1. Fonte Fresco Mobile Cab Package.....	87
2. Fonte Fresco Service MSI	87
3. Fonte Fresco Web Service MSI	87
VII Процес на разработка	89
Заклучение	91
Използвана литература	92

Приложение А Интерфейс на FonteFrescoMobile	93
Приложение В Интерфейс на FonteFrescoService	97

I Увод

С развитието на бизнеса и нарастването на конкуренцията се увеличава необходимостта от софтуерни решения, които максимално да автоматизират и улеснят бизнес процесите във всяка една фирма. Подобни решения могат да се състоят от много компоненти, които са интегрирани помежду си, обменят информация и работят като едно цяло. От особена важност за такива софтуерни решения е възможността за достъп до централизирана информация навсякъде, по всяко време и от различни устройства. Мобилните приложения придобиват все по-голяма популярност, предоставяйки на потребителите си възможност да се интегрират към дадена информационна система, независимо къде се намират, и да обменят информация с нея. Платформата .NET, създадена от Microsoft, обединява различни технологии, чрез които може да се реализира всеки един компонент от информационни системи, улесняващи бизнес процесите във фирма. В настоящата дипломна работа ще бъде представено как са приложени няколко .NET технологии при създаването на информационна система, отговаряща на бизнес нуждите на фирма за монтаж и поддръжка на кафе машини.

1. Цел на дипломната работа

Настоящата дипломна работа има за цел изграждане на информационна система за координиране, регистриране и отчитане на дейността на техници във фирма „Фонте Фреско” ООД. Цели да се постигне моментално и прецизно регистриране и отчитане на дейностите, невъзможни за постигане с досегашната система на ръчна обработка на информацията на хартия и таблици на Microsoft Excel. За съхранение на данните трябва да се проектира и реализира реляционна база данни (Microsoft SQL Server). Информационната система трябва да има две клиентски приложения за двата типа потребители, които я използват – *Администратори* и *Техници*. Достъпът от клиентските приложения до базата данни трябва да се осъществява през XML Web Service.

2. Структура на дипломната работа

В дипломната работа ще бъдат разгледани базата данни на системата и двете клиентски приложения, тъй като са реализирани основно от дипломанта. Дипломната работа се състои от увод, седем части, заключение, списък с използвана литература и приложения.

Част I - „Увод” – запознаване с темата и целите на дипломната работа.

Част I – „Описание на предметната област и потребителските изисквания” – запознаване с дейността на фирмата, участниците и техните дейности, която трябва да се реализират от програмата.

Част II – „Използвани технологии” – кратък преглед на основните компоненти и предимства на технологията .NET и .NET CompactFramework и възможностите, които дават за работа с бази данни.

Част III – „Трислойна архитектура на приложението” – Описание на трислойния модел, функциите на всеки от слоевете и предимствата на модела. Представяне на слоевете от трислойната архитектура, чрез които е реализирано приложението.

Част IV – „Проектиране и реализация на клиентския слой” – Описание на архитектурата на двете клиентски приложения.

Част V – „Реализация на базата данни” – Представяне на логическата структура на базата данни към системата.

Част VI – „Внедряване” – Представяне на създадените инсталационни пакети към всеки модул от системата.

Част VII – „Процес на разработка” – Обосновка за избор на итеративен процес на разработка на системата и описание на приложените практики от Extreme Programming модела.

Заклучение - обобщава резултатите от разработката на дипломната работа.

Използвана литература - представя списък на използваните в процеса на разработка на системата литературни и електронни източници.

Приложение А Потребителски интерфейс на FonteFrescoMobile

Приложение Б Потребителски интерфейс на FonteFrescoService

I Описание на предметната област и потребителските изисквания

Основната дейност на фирма „Фонте Фреско” ООД е свързана с монтаж и поддръжка на кафе машини, разположени в различни обекти: кафенета, магазини, болници и т.н. Обслужващият персонал на фирмата са техници, които извършват определени дейности върху машините или други, които не са свързани с машини. За всяка дейност се отчита определена информация според типа ѝ. Като задължителни данни за всяка дейност трябва да фигурира километраж, при пристигане на обекта където се извършва дейността.

Всяка кафе машина принадлежи към определен оператор и може да се намира в едно от следните състояния:

- нова машина – все още не е монтирана никъде,
- монтирана – машината е монтирана на обект,
- демонтирана – машината е демонтирана от обекта, на който се е намирала, и е в някой от техниците,
- оставена в склад – машината се намира в някой от складовете,
- взета от склад – машината е взета от склада, където е била оставена, и е в някой от техниците,
- неактивна – преустановена е употребата на машината,

Някои от дейностите, които техниците извършват водят до промяна на статуса на машините. Най-общо дейностите свързани с машини се делят на:

- сервизни дейности – монтаж, ремонт, демонтаж, профилактика и авария,
- несервизна дейност,
- посещение в склад – вземане/оставяне на машина.

Други дейности на техниците, които не са свързани с работа по машини са:

- зареждане на гориво,
- посещение на банка,
- служебно друго,
- местодомуване,
- частно посещение.

Някои от дейностите на техниците могат да бъдат зададени от администратори, а други да бъдат инициирани от тях самите. Сервизните дейности например, задължително изискват подадена заявка от администратор.

За улеснение на работата на техниците трябва да бъде създадено Pocket PC приложение, което да отговаря на следните клиентски изисквания:

- Идентификация на потребителят чрез парола и номер на апарат.
- Запазване на последната валидна парола, с която се е идентифицирал потребителят. При липса на връзка със сървъра, въведената парола се валидира спрямо последната валидна парола, запазена в апарата.
- Задаване на настройки на програмата: интернет адрес, време за отговор и клавиатура.

- Получаване на данни за нови оператори, типове машини, типове аварии, адреси на складове и задачи, или обновяване на информацията за стари такива. Това става автоматично на всеки час, когато програмата е стартирана и при успешна идентификация на техник.
- Възможност за съхранение на въведената информация в апарата, при липса на връзка със сървъра(*offline* режим).
- Възможност за преглед на задачите, зададени от администратора, избор на задача, или отказ от избрана задача.
- Въвеждане на детайлна информация за дейност от един от следните типове: Сервизно посещение(Монтаж, Ремонтж, Демонтаж, Профилактика, Авария), Несервизно посещение, Местодомуване, Посещение склад, Посещение банка, Посещение гориво, Служебно друго или Частно посещение в *offline* режим на работа на програмата.
- Въвеждане на детайлна информация за дейност от тип:
 - Сервизно посещение(Монтаж, Ремонтж, Демонтаж, Профилактика, Авария) и Несервизно посещение в *online* режим на работа на програмата, само ако потребителят има получена задача за такъв тип дейност.
 - Сервизно посещение(Профилактика), Местодомуване, Посещение склад, Посещение банка, Посещение гориво, Служебно друго или Частно посещение в *online* режим на работа на програмата.
- Преглед на записи, които са били съхранени в телефона по време на *offline* режима на работа на програмата и възможност за или смяна на задачата с която са били асоциирани(ако има такава).
- Възможност за изтриване на запис, който е бил съхранен в телефона:
 - Преди да бъде детайлно прегледан записа.
 - След като бъде детайлно прегледан записа.
 - При валидация на номера на машината, за която се отнася записа. Ако се окаже, че номера не е валиден, потребителят има възможност да изтрие записа.
- Възможност за редактиране на въведената информация чрез връщане назад по формите.
- Валидация на въведената информация при преминаване към следващата форма.
- Валидация на номера на машина при въвеждане на данни за дейност, свързана с тази машина. Трябва да се провери не само дали съществува такава машина, но и дали статусът и позволява да се въведе запис за избраната от техника дейност.
- Проверка на батерия, налична памет и дисково пространство при стартиране на програмата, при всяко преминаване към следваща форма и при всяко обръщение към сървъра.

- Информирание на потребителят за отмяна на задачата, върху която работи, когато е в *online* режим на работа и задачите му са били обновени, след като администратора е отменил задачата.
- Възможност за преминаване по контроли и указване на избор в някои от тях със PDA Soft key бутон.
- Автоматично показване на клавиатурата, указана в настройките на програмата, при попадане в текстово поле.

За работата на администраторите трябва да се създаде Windows Forms приложение, което да отговаря на следните изисквания:

- Да бъде MDI(Multiple Document Interface) приложение.
- Използване на програмата само от потребители, регистрирани като администратори.
- Възможност за използване на програмата от няколко администратора едновременно.
- Задаване на настройки на програмата: интернет адрес, време за отговор, свързване през прокси, адрес и порт на прокси.
- Идентификация на администраторите чрез парола.
- Контролиране на достъпа на техници до системата от Pocket PC приложението.
- Програмата трябва да предоставя възможност за съхранение и обработка на следната информация:
 - Потребители - въвеждане на нов техник/администратор, редакция на данни за техници/администратори, извличане на справка за техници/администратори.
 - Задачи - въвеждане на нова задача, редакция на не приключени задачи, извличане на справка за приключени задачи.
 - Обекти - въвеждане нов обект, редакция на данни за обект, извличане на справка за обекти.
 - Типове машини - въвеждане на нов тип машина, редакция съществуващи типове машини, извличане на справка за типове машини.
 - Машини - въвеждане на нова машина, редакция на данни за данни за машини, извличане на справка за машини.
 - Оператори - въвеждане на данни за нов оператор, редакция на данни за оператори, извличане на справка за оператори.
 - Складове - въвеждане на нов склад, редакция на данни за складове, извличане на справка за складове.
 - Типове аварии - въвеждане на нов тип авария, редакция на типове аварии, извличане на справка за типове аварии.
- Анулиране на задача само ако не е приключена все още от техник.
- При деактивиране на оператор, автоматично деактивиране и на неговите складове.
- Получаване на детайлни справки:

- Справка за направените профилактики – детайлна и обща
 - Справка за направените монтажи групирани по тип на машина, оператор и месец
 - Справка за направените Сервизни дейности(монтаж, ремонт, демонтаж, профилактика и авария) групирани по тип на сервизна дейност, оператор и месец
 - Справка за направените Сервизни дейности(монтаж, ремонт, демонтаж, профилактика и авария) групирани по тип на сервизна дейност, техник и месец
 - Справка за пътен лист за всеки техник за всеки ден със заредено гориво
 - Справка броя дейностите извършени от всеки техник групирани по вид дейност
 - Справка за дейностите извършени от всеки техник и времето между дейностите
 - Справка за приходи и разходи
 - Справка за всички дейности извършени върху определена машина
 - и др.
- Филтриране на справки по различни критерии.

II Използвани технологии

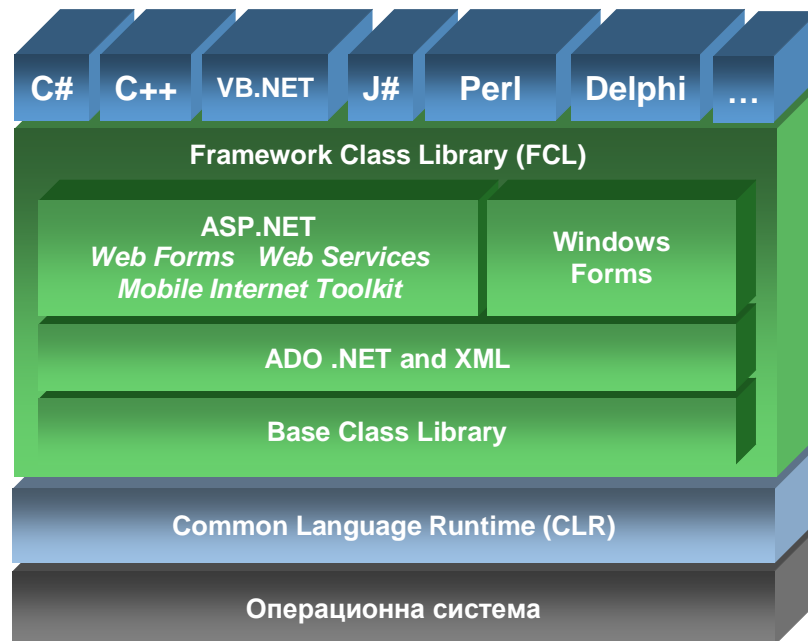
1. NET Framework

.NET е платформа на Microsoft, създадена с висока интеграция за XML Web Services. Някои от ключовите моменти при дефинирането и са:

- .NET е платформа за разработка и разпространение на модерни обектно – ориентирани, „управлявани” приложения.
- Функционални приложения за .NET могат да бъдат разработени на всеки един програмен език предназначен за .NET runtime.
- .NET се състои от библиотеки от класове, независими от програмен език, поддържа създаването на себе описващи софтуерни компоненти, единна инфраструктура за разработка, независима от езиците за програмиране, преизползване и наследяване на компонентите от едни езици в други.
- .NET предлага един нов начин на писане на десктоп приложения за Windows, използвайки Windows Forms класове, нов начин на писане на Web-базирани приложения, използвайки ASP.NET класове и една нова „несвързана” архитектура за достъп до данни през интернет, използвайки ADO.NET класове.
- Поддържа създаване на платформено-независими XML Web Services, използвайки стандарти като SOAP (Simple Object Access Protocol) и WSDL (Web Service Description Language)

.NET управлява всеки един аспект от работата на приложенията. Управлява паметта за данни и инструкции, оторизира приложението спрямо подходящите права, инициира и управлява стартирането на програмите и освобождава заетата памет от ресурси, които не са необходими.

.NET Framework се състои от два основни компонента – CLR (Common Language Runtime) и FCL (Framework Class Library)



фиг. 1 Архитектура на .NET Framework

1.1. Common Language Runtime

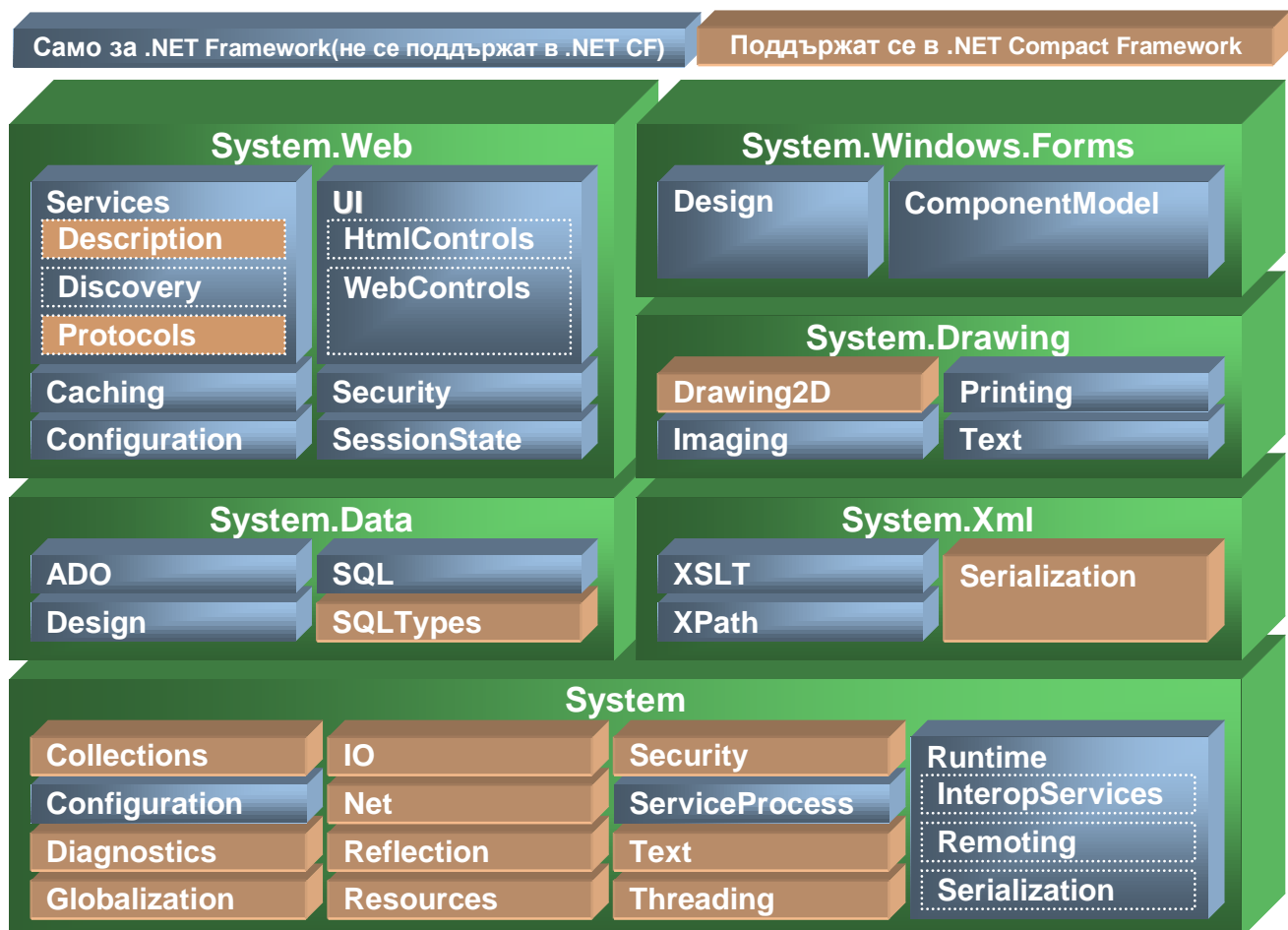
CLR може да се разглежда като средата, която управлява изпълнението на кода на приложението. Тя предоставя базови услуги като компилация на кода, заделяне на памет, управление на нишки, система за почистване на паметта (*garbage collection*). CLR се намира „над“ операционната система. Когато се стартира „управлявано“ приложение CLR го зарежда и изпълнява кода му. Кодът на такова приложение се нарича „управляван“ код и се състои от инструкции във вид на псевдо-машинен език наречен CIL (*Common Intermediate Language*). Тези инструкции се компилират до машинен език по време на изпълнение за съответната хардуерна архитектура (обикновено x86). Този процес се нарича JIT (*just-in-time*) компилиране. Обикновено даден метод се компилира само веднъж, когато се извика за първи път и след това се съхранява в паметта. JIT компилацията не оказва драстично отрицателно влияние на производителността поради факта, че се извършва само веднъж през целия живот на приложението. На теория дори може да се подобри производителността на вече съществуващ код поради факта, че JIT компилатора оптимизира кода, който генерира за текущата версия на процесора, на който се изпълнява приложението. JIT компилирания код не е същият както интерпретирания код. Това е и една от основните разлики на .NET от JAVA. Много са предимствата на „управлявания“ код. По време на компилация CLR проверява кода, който ще бъде изпълнен, за грешки. Не е възможно да се извика инструкция, която се обръща към памет, към която инструкцията няма права за обръщение, CLR ще предизвика изключение при тези случаи. В допълнение към елиминирането на тези и други често срещани грешки, е увеличена сигурността при изпълнение на програмите. Системата проверява всеки код преди да се изпълни. Тази система за проверка на сигурността на кода позволява да се създадат и виртуални участъци във рамките на един процес, наречени *application domains*. Принципът им е изолация както при отделните процеси в операционната система, но във рамките на процеса, в който съществуват. Така приложенията при многопотребителски системи се

създават в *application domain*, а не се създава отделен процес за всеки потребител. (Основни примери са ASP.NET и Web Services). Един процес може да host-ва множество *application domains* и заредените библиотеки могат да се използват едновременно от всички приложения.

Друго предимство на „управлявания“ код е системата за освобождаване на неизползвана памет. CLR включва сложен *Garbage Collector* (GC), който следи за референции към обекти в кода и освобождава паметта като унищожава обекти, които вече не се използват никъде в програмата. GC подобрява производителността, защото алгоритъма използван от CLR е много по – бърз от еквивалентните функции за заделяне на памет от C runtime. Има и негативна страна. Когато се стартира този алгоритъм, всяка нишка в процеса е спряна до приключване на *garbage collection*.

1.2. Framework Class Library

FCL е библиотека от повече от 7000 типа класове, структури, интерфейси, списъци и делегати (строго типизирани референции към callback функции). FCL е разделена на йерархични пакети. Всеки пакет съдържа класове и типове споделящи обща цел. Физически FCL се съдържа в съвкупност от DLL файлове в директорията `%SystemRoot%\Microsoft.NET\Framework\v1.n.nnnn`. Всеки DLL файл е асембли, което може да се зареди от CRL при нужда. Основните типове данни като Int32 са реализирани в `mSCORELIB.dll`, други са разпръснати по FCL DLL файловете.



фиг. 2 Пакетите от библиотеката FCL

Важна част от FCL е пакетът System.Data. В него са реализирани ADO.NET компонентите за работа с бази данни. ADO.NET представлява революция спрямо предишните версии на ADO. От самото си начало ADO.NET е конструиран да работи в „несвързаната“ среда на Web пространството, да запълни празнината между релационните данни и XML чрез тясна интегрираност между тях, да предостави общо представяне на данните комбинирани от различни източници за данни и оптимизиран достъп до бази данни. Въпреки това програмния модел цели да остане максимално съвместим с предишните версии на ADO. ADO.NET предоставя отлични възможности за разработка на приложения с многослойна архитектура.

ADO.NET се състои от два основни компонента: *DataSet* и *Data Provider*. Последния е съвкупност от няколко важни компонента – *Connection*, *Command*, *DataReader* и *DataAdapter*.

1. *DataSet* компонент

Ключов компонент на „несвързаната“ (disconnected) архитектура на ADO.NET. Той е изрично проектиран да предоставя достъп до данни, независимо от източника им. Може да съдържа данни локални за програмата, XML данни, и от други източници. Той съдържа списъци от един или няколко *DataTable* обекти съставени от редове и колони, информация за първични, вторични ключове, ограниченията и релационна информация за данните в *DataTable* обектите. С други думи *DataSet* е изглед на база данни в паметта на компютъра. Той е винаги „несвързан“ и няма информация за източника на данните. *DataSet* е идеален за съхранение на резултати от заявки към бази данни, редактиране на данни и съхранение на данните, обратно в източника и за кеширане при Web приложенията. *DataSet* обектите притежават няколко метода, които допринасят за тясната интеграция с XML - *WriteXML()*, *WriteXMLSchema()*, *ReadXML()*, *ReadXMLSchema()*. Чрез тях цялото съдържание на *DataSet* може да се запише и прочете като XML данни. Това включва както данните от таблиците така и релационните данни, първични и вторични ключове, ограничения.

Поради факта, че *DataSet* е „несвързан“ с източника на данни, той трябва да има механизми за следене на направените промените. Това са *HasChanges()*, *GetChanges()* и *Merge()* методи. *DataSet* може да се конструира в паметта без да е необходимо да има външен източник на данни, но на практика винаги се запълва с данни от резултати от заявки към база данни или от XML документи. Компонентът не „общува“ директно с базата данни, използва *DataAdapter* обект. Той извършва заявките и създава *DataTable* обекти с резултатите от тях. Той извършва и записа на променените данни обратно в базата данни. Явява се като ниво на абстракция между *DataSet* обекта и физическата база данни. *DataSet* обекта е само описващ се чрез своята схема и данните. Данните, които съдържа могат да се представят с чист XML и може да бъде пренесен със стандартни протоколи през мрежова среда към други програми и софтуерни модули.

2. *Data Provider* - (Доставчици на данни)

ADO.NET има разделена същност що се отнася до достъпа до бази данни. Той се осъществява чрез софтуерни модули, наречени доставчици на данни (*Data Providers*). В версия 2.0 на .NET Framework са включени четири доставчика:

- SQL Server .NET доставчик, който се обръща към Microsoft SQL Server бази данни с напълно „управляван” код.
- OLE DB .NET доставчик, който ползва помощ от „неуправлявани” софтуерни модули на OLE DB технологията.
- ODBC .NET доставчик, използва „неуправлявани” ODBC Driver Management (DM) през COM *interop* за да осигури достъп до ODBC източник на данни.(не е включен в .NET 1.0).
- .NET доставчик за Oracle, позволява достъп до Oracle база данни през Oracle client connectivity software не е включен в .NET 1.0).

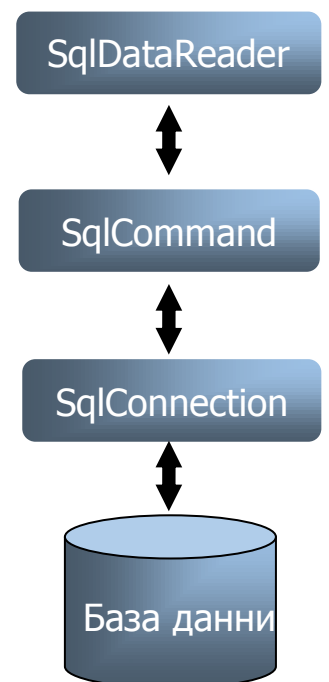
Изборът, който доставчик ще използва зависи от конкретната архитектура на приложението и използваната система за управление на бази данни. Някои класове работят със всички доставчици. Такъв е *DataSet*. Други са дефинирани строго за определен вид доставчик. Например *DataAdapter* има за SQL Server .NET Provider (*SQLDataAdapter*), за OLE DB .NET Provider(*OleDbDataAdapter*), за ODBC .NET Provider(*OdbcDataAdapter*) и за Oracle .NET Provider(*OracleDataAdapter*). Това разграничаване не е фатално, защото и четирите типа класове използват едни и същи методи и събития.

a) *Connection* – компоненти

Има четири вида – *OleDbConnection*, *SqlConnection*, *OdbcConnection* и *OracleConnection*. И четирите компонента реализират *System.Data.IDbConnection* и наследяват свойства като *ConnectionString* и *ConnectionState* и притежават методи *Open* и *Close* за инициране на връзка и приключване на такава. Приликите със старата версия ADO *Connection* обект са много, но има и разлики. *Connection* обект във ADO.NET не реализира поддръжка на транзакции. Тя е отделена във друг тип обекти *OleDbTransaction*, *SqlTransaction*, *OdbcTransaction* и *OracleTransaction*. Няма я и реализацията на изпълнение на SQL заявки, която е реализирана в отделен обект *Command*. Това е добър начин за постигане на разделение на функционалността между класовете.

b) *Command* - компоненти

Функционалността на *SqlCommand*, *OleDbCommand*, *OdbcCommand* и *OracleCommand* обектите е идентична с тази на ADO *Command* обектите. Те позволяват да се изпълни заявка към машината за база данни за добавяне, промяна и изтриване на данни (*select*, *insert*, *update*, *delete*). Всички обекти са свързани с един *Connection* обект и изпълняват заявките си през него. За да се изпълни заявка трябва *Connection* обекта да е иницирал връзка към машината за бази данни. Една от големите разлики с ADO, е че резултата от заявка не е *recordset*, а обект от тип



фиг. 3
ADO.NET в свързана среда

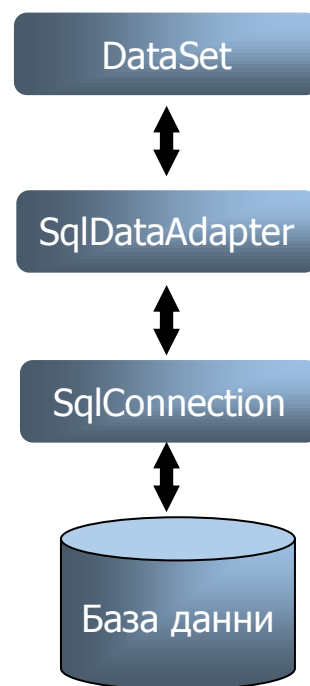
DataReader, чрез който могат да се прочетат данните от резултата.

c) *DataReader* – компоненти

Те представляват една нова концепция спрямо ADO. Може да се разглежда като поток от данни (stream). Това е *forward-only* курсор от страна на сървъра и връзката със сървъра остава постоянна докато се четат данните. Поради това *DataReader* обект се използва, когато се реализира свързана среда. (фиг.3)

d) *DataAdapter* – контроли

ADO.NET въвежда и друг вид обекти – *DataAdapter*. Тези обекти служат като мост между източника на данни и „несвързания“ *DataSet*. Тези обекти са също свързани с *Connection* обект, чрез който се прави физическата връзка към източника на данни. Освен това *DataAdapter* обектите имат набор от команди за извличане на данни от източника и съхранението им в една *DataTable* на обекта *DataSet*, както и команди за запис на промените, които са направени върху текущите данни от *DataSet* обекта. Командите за извличане на данни са SQL *select* заявки под формата на свойство *SelectCommand*, а тези за обновяване на данни са SQL *insert*, *update*, *delete* под формата съответно на свойства *InsertCommand*, *UpdateCommand* и *DeleteCommand*. Два от най-важните методи са *DataAdapter.Fill()* и *DataAdapter.Update()*. *Fill()* метода използва *SelectCommand* за да извлече данни от източника на данни, а *Update* метода използва една от трите SQL заявки *insert,update,delete* за да добави, промени или изтрие записи в зависимост от направените промени в *DataSet*. Важно свойство е *TableMappings*. То е списък от *TableMapping* обекти, чрез които може да се съпоставят имена на колони от таблиците с по – приветливи имена. Това още повече изолира *DataSet* обекта от физическия източник на данните



фиг. 4
ADO.NET в
несвързана среда

2. .NET Compact Framework

.NET Compact Framework е подмножество на .NET Framework. То се състои от основните библиотеки с класове и има няколко допълнителни библиотеки, които са специфични за разработката на приложения за мобилни устройства. Common Language Runtime (CLR) е написана отначало, за да бъде специфична за .NET Compact Framework така че тя се стартира по-ефективно на мобилни устройства, които са ограничени в паметта, ресурсите, и трябва да щадят батерията. За да е възможно да се стартира .NET Compact Framework приложение, платформите трябва да поддържат .NET Compact Framework runtime. .NET Compact Framework изисква минимум Windows CE .NET

(Windows CE версия 4.1 за released версия на .NET Compact Framework) с две изключения: Microsoft Pocket PC, Microsoft Pocket PC 2002, и Microsoft Pocket PC 2002 Phone Edition поддържат .NET Compact Framework. SmartPhone 2002 не поддържа .NET Compact Framework; въпреки че SmartPhone 2003 поддържа.

2.1. Предимства и цели

- Споделен код и повишаване на продуктивност.
- Сигурност при изпълнение.
- Автоматично разпространение(*deployment*).
- *Rapid* разработка на приложения.
- Единно бинарно разпространение(*deployment*), което се стартира върху различни процесори, върху една и съща платформа без прекомпилиране.
- Тъй като прекомпилирането не е необходимо чрез .NET Compact Framework, контролите, приложенията и услугите могат лесно да се преместват от едно устройство на друго.
- Възможност за извикване на Win32 DLLs без да е необходимо да се пренаписват.
- Преносимо подмножество на .NET Framework.
- Дебъгването и разработката може да се извършва с Visual Studio.NET.
- Кода не е податлив на сринове, произхождащи от не инициализирани променливи.
- Системата за почистване на паметта(*garbage collection*) значително минимизира загубата на памет, освобождавайки паметта, заемана от обектите, които повече не се използват.
- Използване на същата конвенция за именуване както в .NET Framework.
- Поддръжка на Simple Object Access Protocol(SOAP).

2.2. SQL CE

Microsoft SQL Server CE е компактна, релационна база данни която се стартира на Smart Devices. SQL Server CE има вградена поддръжка на .NET Compact Framework, използва съвместими типове данни. SQL Server CE може да се инсталира независимо на smart devices или като част от .NET Compact Framework приложения. SQL Server CE поддържа бази данни до 2GB. Също поддържа Binary Large Objects(BLOBS) до 1GB. SQL Server CE е идеален механизъм за съхранение на данни поради възможностите си за криптиране.

SQL Server CE се състои от database engine, client agent, и много малка версия на Query Analyzer. SQL Server CE поддържа подмножество на Transact SQL. SQL Server CE може да съхранява данни в релационен формат, извлича тези данни, и ги доставя като XML документ за употреба при Web базирани приложения.

За създаване на приложения чрез .NET Compact Framework, трябва да се използва ADO.NET управляван доставчик (System.Data.SqlServerCE). Този клас предоставя

достъп до данни съхранявани директно в базата данни използвайки Remote Data Access(RDA) и/или *merge of replication*. Използвайки *merge replication*, може да си осигурите че данните между сървъра и базата данни на клиента са консистентни.

2.3. Разлики с .NET Framework

.NET Compact Framework имплементира приблизително 30% от цялата библиотека с класове на .NET Framework и също съдържа свойства и класове специфични за разработката на мобилните приложения.

- *Application Domains* – не се поддържа в .NET Compact Framework.
- *ASP .NET* - .NET Compact Framework е основно платформа за *rich* клиенти и не предоставя поддръжка на ASP. NET.
- *Assemblies and Global Assembly Cache* - .NET Compact Framework не поддържа асемблита *multi-module* асемблита, но поддържат *satellite* асемблита.
- *Classes and Types* – .NET Compact Framework поддържа подмножество от библиотеката с класове на .NET Framework. Класовете които се поддържат могат да се видят на фиг 2.
- *Native Code InterOP* – във версия 1.0 на .NET Compact Framework се поддържа само P/Invoke функциите и атрибутите от System.Runtime.InteropServices. Във версия 2.0 се поддържа и COM interloper.
- *CLR* – CLR и в двата Framework-а се възползва от изпълнение на управляван код, *just-in-time*(JIT) компилация на код, и *garbage collection*. CLR за .NET Compact Framework е приблизително 12% от CLR на целия .NET Framework.
- *Controls* - .NET Compact Framework поддържа повечето от Windows Forms контролите от .NET Framework и добавя някои специфични.
- *Контроли, които не се поддържат:* GroupBox, RichTextBox, NotificationBubble, CheckedListBox, ColorDialog, ErrorProvider, HelpProvider, LinkLabel, NotifyIcon, ToolTip, Splitter, FontDialog
- *Контроли, които са добавени:* Clipboard(v2.0), DocumentList(v2.0), InputPanel, InputMethodCollection(v2.0), InputMode(v2.0), MobileDevice(v2.0), ScreenOrientation(v2.0), WindowsMessage.
- В .NET Compact Framework няма поддръжка на drag and drop, на принтиране, няма поддръжка на Microsoft ActiveX®, и на GDI+.
- *Current Directory* – Тази функционалност не е предоставена в Windows CE операционната система, така че .NET Compact Framework не поддържа *GetCurrentDirectory* и *SetCurrentDirectory* методите.
- *Data – System.Data.OleDb* пространството от имена не се поддържа.
- *Deploying Applications* – Разпространението е толкова лесно, колкото да се копира асемблито на устройството чрез кабел от компютъра, инфрачервен, безжична интернет връзка или чрез интернет връзка. В MS Visual Studio

2005 приложенията могат да се разпространяват директно на устройството по време на debug.

- *Disposed Objects* – Достъпването на методи или свойства на *disposed* обекти винаги ще са неуспешни в .NET Compact Framework.
- *Globalization* – Свойството *CurrentCulture* на класа *CurrentThread* не се поддържа в .NET Compact Framework, може да се използва променливата *CultureInfo*.
- *Events* - .NET Compact Framework поддържа *GotFocus* и *LostFocus* събитията, но не поддържа *Activate* и *Deactivate*.
- *Exception Description Strings* - .NET Compact Framework предоставя съобщението при възникване на изключение в отделен DLL, System.SR.DLL, за спестяване на памет.
- *Input/Output* - Поради различията в операционните системи; има ограничения на I/O моделите. .NET Compact Framework не предоставя информирани при промяна на файл.
- *Math* – Не всички математически методи се поддържат на всички платформи; въпреки това те са включени в API поради съвместимост.
- *Networking* - .NET Compact Framework предоставя Infrared Data Association(IrDA) класове за осъществяване на инфрачервен връзки и Web listening класове за обслужване на HTTP заявките към устройството. Тези класове са налични единствено в .NET Compact Framework.
- *nGen (Install-time JIT)* – Поддържа се в .NET Framework но не и в .NET Compact Framework.
- *Proxy Code* - .NET Compact Framework не поддържа целия код генериран от Web Services Description Language Tool(Wsdl.exe).
- *Reflection* - .NET Compact Framework не поддържа *System.Reflection.Emit* пространството от имена. Също не поддържа оператора за равенство(==) при сравняване на *reflection* обекти като *MethodInfo*, *FieldInfo*, *PropertyInfo*, *EventInfo*, *MemberInfo*, *MethodBase*, *ConstructorInfo* и *ParameterInfo*.
- *Remoting* – Няма поддръжка на .NET Remoting в .NET Compact Framework. XML услугите се използват като алтернатива за това.
- *Secure Messaging* - .NET Compact Framework не поддържа сертифициране и автентикация от страна на клиента чрез HTTPS.
- *Serialization* - Поради ограниченията за производителност, .NET Compact Framework не поддържа бинарна сериализация използвайки *BinaryFormatter*, или SOAP сериализация използвайки *SoapFormatter*. Въпреки това поддържа сериализация за пренасяне на данни за обекти използвайки SOAP в XML Web Services.
- *Sockets* – Не се поддържат всички опции на сокетите от .NET Compact Framework.
- *String Manipulations, Regular Expressions* – Приложенията, които използват регулярни изрази в .NET Compact Framework не са бинарно съвместими с приложенията в .NET Framework, но техните сорс код е съвместим.

- *Time Intervals* – Стойността върната от *Now* е представена само в секунди, не в милисекунди. Може да се вземе по точно измерване чрез *TickCount* свойството.
- *Timers* – *Start* и *Stop* методите на *System.Timers.Timer* не се поддържат, но може да се стартира и да се спре отброяването чрез *Enable* свойството на *System.Windows.Forms.Timer*.
- *Web Services* - – Само клиентска имплементация се поддържа в .NET Compact Framework. Има някои ограничения за .NET Compact Framework. Няма типизирани datasets, и няма поддръжка на бинарна сериализация или SOAP сериализация(все пак се поддържа сериализация на обекти).
- *XML* - Налична е намелена поддръжка на XML. .NET Compact Framework приложенията имат пълна поддръжка за четене и писане на XML използвайки определени класове за тази цел, но те нямат поддръжка за XSD, XSLT или XPATH.

2.4. Стратегии за достъп до данни в .NET Compact Framework

Стратегиите за достъп до данни в .NET Compact Framework се отнасят към няколко аспекта. Ключовият аспект, от който всички други зависят, е свързаността. Тъй като Pocket PC може да се използва и *online* и *offline*, стратегиите за достъп до данни е необходимо да определи как се:

- управляват и използват данни от отдалечен сървър при *online*.
- съхраняват и използват данни при *offline*.
- разменят данни когато Pocket PC стане *online* от предишното *offline* състояние.

Основно, стратегиите за достъп до данни в .NET Compact Framework се влияят от два аспекта:

- ✓ Как да се съхраняват данните на Pocket PC - От гледна точка на приложението, данните могат да съществуват в релационна база данни(като Microsoft SQL Server CE), в локален файл(като XML файл често управляван чрез *DataSet*), и в базирана на сесии структура данни, в паметта, която се освобождава, когато приложението приключи.
- ✓ Как да се разменят данни със сървъра - Разменянето на данни между сървъра и Pocket PC може да се имплементира различно в зависимост от това какъв Pocket PC слой с какъв слой от сървъра комуникира. Възможностите за размяне на данни характерни за .NET Compact Framework са:
 - Pocket PC база данни към база данни на сървъра: Базата данни на Pocket PC разменя данни директно със базата данни на сървъра. Тази възможност е валидна в сценарии, в които има малко или няма бизнес логика и в сценарии с голям брой данни. Имплементира се чрез *Remote Data Access* и *Merge Replication* свойствата на SQL Server CE. Ако се изисква синхронизация на данните тогава *Merge Replication* предоставя

вградено разрешаване на конфликтите имплементирано в SQL Server Reconciler на сървъра.

- Pocket PC компонент към сървър компонент: Pocket PC приложението комуникира с компоненти на сървъра. Тази възможност позволява бизнес логика, имплементирана чрез Web услуги, да бъде част от размяната на данни. Тази възможност може да се използва и в двата случая, когато данните се съхраняват в локален XML файл или в SQL Server CE база данни.
- Pocket PC компонент към база данни на сървъра: Pocket PC се свързва директно към базата данни на сървъра. Тази възможност често се използва, когато е необходимо Pocket PC приложението да управлява голямо количество данни в отдалечена база данни без да е необходимо да съдържа данните в Pocket PC и когато бизнес логиката не е от основно значение. Тази възможност се имплементира използвайки *System.Data.SqlClient* пространството от имена и може да се използва и в двата случая, когато данните се съхраняват в локален XML файл или в SQL Server CE база данни.
- Само сървър: Елементите от потребителския интерфейс на Pocket PC приложението могат директно да се свързват към компоненти на сървъра или могат да се имплементират като web приложения използващи Pocket PC Web брауъра. Това очевидно изисква Pocket PC да бъде винаги свързано със сървъра. Тази възможност не изисква локално съхранение на никакви данни.

Преди да се вземе решение как да се съхраняват и разменят данните трябва да се обърне внимание на : характеристиките на данните, интернет свързаността и системната архитектура.

Характеристика на данните

Характеристиките на данните се определят следните ключови данни:

1. Количеството на статичните данни и предаващите се данни, което трябва да се съхранява на Pocket PC: Ако количеството на данните е малко, по малко от 50-100 kb, тогава данните могат да се съхраняват в локален XML файл. Ако количеството на данните е по-голямо, тогава SQL Server CE би предоставил по-добра производителност и сигурност. Причината за това е че данните в такъв случай ще се достъпват чрез SQL Server CE Query Engine с поддръжката на SQL което води до по-добра производителност и управляемост.
2. Количеството на данните, обменяно със сървъра: Ако количеството на данните, обменяни между сървъра и Pocket PC е малко, например по-малко от 500kb – 1Mb, тогава данните могат да се предават чрез XML, използвайки Web услуги. Ако количеството данни е по-голямо, *Remote Data Access* и *Merge Replication* ще доведат до по-добра производителност. Причина за това е че SQL Server CE Client и Server Agents имплементират ефектива

компресия на данни и така данните трябва да преминат сравнително по-малък брой операции докато достигнат крайната точка.

Интернет свързаност

Аспекта за интернет свързаността се отнася до това кога Pocket PC ще е *online* и колко често ще е *offline*.

Поради същността на XML, който често съдържа голямо количество от стандартни мета данни без компресия, при Web услугите се изпращат повече данни за предаване на същите „полезни” данни в сравнение с SQL Server CE *Remote Data Access* and *Merge Replication*. Също тези два метода често се използват когато честотата на размяна на данни е малка, което увеличава вероятността че количеството данни, което е необходимо да се обмени е голямо. Това означава, че в реалност, SQL Server CE *Remote Data Access* и *Merge Replication* често се използват при голям обмен на данни, а Web услугите при случаи с малък обмен на данни. Въпреки това, когато се използват Web услуги, се предпочитат по-малки заявки пред по-големи заявки.

Системна архитектура

Мобилните приложения могат да се разглеждат като самостоятелни приложения, свързващи се към съществуващи системи или разширения на съществуващи системи. Pocket PC приложенията най-често не са самостоятелни и трябва да комуникират с други системи.

Днес повечето системи се имплементират като многослойни приложения. Server-side приложенията се изграждат чрез компоненти поради възможността за преизползване на код, разделяне на бизнес логиката от данните и за увеличение на податливостта(*manageability*). Употребата на XML и Web услуги са предпочитани при такива разработки, и .NET Compact Framework приложенията се вписват правилно в тази архитектура поради поддръжката на XML и Web услуги.

3. Web услуги

В съвременните системи все по-често цялата бизнес логика на приложението се изнася в Web услуги. Това е така, защото Web услугите са лесно достъпни през Интернет и осигуряват възможност за между платформена комуникация, т. е. техните консуматори може да са много различни: Web приложения, Windows Forms клиенти, мобилни устройства, както и др. Web услугите отварят системата към взаимодействие с различни крайни клиенти, реализирани върху различни платформи.

XML Web услугите могат да се дефинират като софтуерна услуга достъпни през Web чрез SOAP, описвани чрез WSDL и регистрирани в UDDI.

Архитектурно Web услугите са функционално независими компоненти и са слабо обвързани с клиента, който ги използва (*loosely coupled*). Те използват утвърдения в Интернет и при Web технологиите модел "заявка/отговор" (*request/response*), т. е. за всяка една отделна заявка към сървър, той връща отделен отговор специално за нея.

- SOAP (Simple Object Access Protocol) – SOAP е XML базиран стандарт за обмяна на структурирана и типизирана информация в Web пространството. Използва някои утвърдени стандарти: XML за описания на съобщения, XSD за описание на използваните типове и HTTP като транспортен протокол. Структурата на SOAP съобщенията не е сложна. Те се състоят от хедър (SOAP header) и тяло (SOAP body) като SOAP хедъра не е задължителен.
- WSDL (Web Service Description Language) – за описание на интерфейса и начина на достъп до Web услугите се използва XML базираният език WSDL. Използвайки XML и XSD, WSDL изгражда едно абстрактно описание на услугата – поддържани методи, използвани типове данни, начин на достъп до услугата. Това означава, че WSDL дефинира всичко необходимо за да се създаде програма, която да работи с XML Web услугата.
- UDDI(Universal Description, Discovery, and Integration) – публикуването и откриването на информация за дадена Web услуга става посредством UDDI стандарта. Той е отворен XML базиран стандарт за регистриране, откриване и свързване към Web услуга. UDDI сам по себе си също е web услуга. Нейната функционалност включва регистрация и търсене на други услуги.

III Трислойна архитектура на приложението

1. Трислоен модел на комуникация

Чрез появата на Local-Area-Networks, персоналните компютри излезли от своята изолация. Така се зародила Клиент/Сървър комуникацията. Клиент/Сървър е архитектурен модел на системи за доставяне на информация на крайния потребител. Типичен случай на такава система е сървър с база от данни и множество клиенти, които работят с общите данни от сървъра.

За съжаление двуслойният модел демонстрира голяма слабост, така че създаването и поддръжката на такива приложения много се оскъпява. Едни от недостатъците му са:

- Завършените приложения се натрупва на компютрите.
- В двуслойната архитектура бизнес логиката се осъществява на компютъра на клиента.
- Windows 3.X и Mac системите имат строги ограничения на ресурсите. По тази причина приложните програмисти също трябва да бъдат добре запознати със системните технологии.
- РС-тата се разглеждат като „untrusted” в смисъла на сигурност.
- Приложната логика не може да бъде повторно използвана, защото тя е прикрепена към индивидуална РС програма.

Поради тези и други причини се налага изграждането на трислойни и многослойни разпределени приложения. Трислойната и многослойната архитектури се стремят към решаване на недостатъците на двуслойната архитектура. При трислойната архитектура се добавя междинен слой, който да изпълнява бизнес логика и контролира достъпа до базата данни. По този начин клиентския слой става независим от бизнес логиката. Така при промени в бизнес логиката приложенията работещи при клиентите няма да се променят. В трислойният модел, клиента изпълнява представителната логика, бизнес логиката се осъществява на приложния сървър, а данните се съхраняват на слоя за управление на данни.

1.1. Функции на слоевете.

- **Представителен слой**

Той отговаря за представянето на данните, получаване на събития от потребителя и контролиране на потребителския интерфейс. Всъщност бизнес логиката се премества на приложния сървър.

- **Бизнес слой**

Този слой формира основния ключ за решаване на проблемите на двуслойната архитектура. Той защитава данните от директен достъп от клиентите.

Сървъра на средното ниво, също относящ се към бизнес слоя, подобрява изпълнението, гъвкавостта, поддръжката, много използваемостта чрез централизирана логика на процесите. В допълнение, управлението на процесите от средното ниво

контролират транзакциите и асинхронното извикване за да осигурят надеждно изпълнение на транзакциите, добавя планиране и приоритетност на работите в процеса.

- **Сървър за данни**

Този слой отговаря за съхранението на данните.

1.2. Предимства на трислойната архитектура

- Ясно изолиране на контрола на потребителския интерфейс и представянето на данните от приложната логика. Двете важни предимства на клиентското приложение са ясни: по-бързо създаване чрез много използвани компоненти на бизнес логиката и по-кратка фаза на тестване, защото сървърните компоненти вече са били тествани.

- Клиента е изолиран от базата данни и мрежовите операции.

- В сравнение с двуслойния модел, където само данните са публично достъпни, бизнес обектите могат да предоставят приложната логика или „услугите” в мрежата.

- Сървърите могат да стартират критични бизнес процеси, които работят с данни изискващи защита.

- Задръстването на мрежата се намаля, защото приложния слой не обменя извънредно много данни към клиентите, а само това което е необходимо да се съхранява на диска.

- Ако настъпи безизходно положение по време на изпълнение, сървърните процеси могат да се преместят на друг сървър по време на изпълнение.

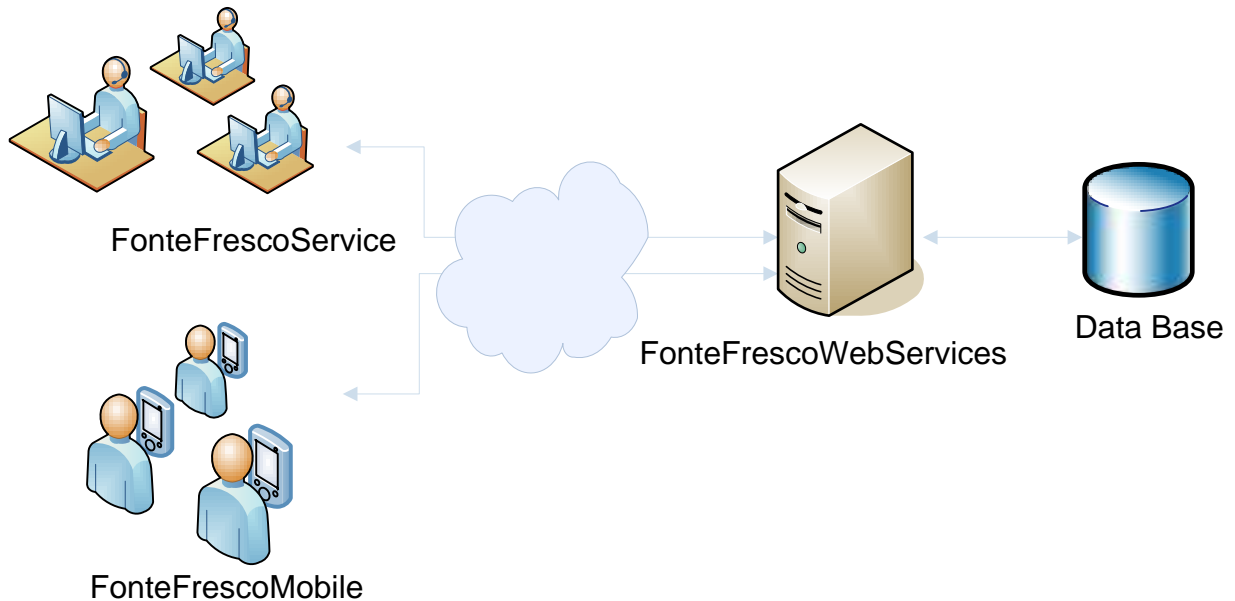
- Улеснява софтуерното проектиране, защото отделните слоеве могат да се създават и изпълняват на отделни платформи, това я прави лесна за организиране на изпълнението.

- Допуска различните слоеве да са създадени на различни езици.

- Когато бизнес логиката се налага да претърпи промени само сървъра трябва да се обновява. В двуслойната архитектура всеки клиент трябва да се модифицира, когато се сменя логиката. Нова функционалност може да бъде добавена лесно без да се модифицира съществуващата функционалност.

2. Трислойна архитектура на приложението

Информационната система е реализирана чрез трислойна архитектура(фиг. 5).



фиг. 5 Схема на трислойна архитектура на системата

2.1. Представителен слой

Клиентския слой се реализира с приложенията :

- *FonteFrescoService* – Windows Forms приложение, работещо на MS Windows. Изисква инсталиран .NET Framework v2.0 на всеки компютър. То съдържа множество от форми, логически отделени на четири групи в зависимост от функциите, които се предоставят на потребителите на програмата:

- а) Форма за вход/ изход в системата, извършва идентификация на администраторите.
- б) Форми за добавяне на данни за: потребители, машини, оператори, складове, обекти, задачи, типове машини и типове аварии. Във всички форми се извършва валидация на въведените данни и асинхронно обръщане към Web методи за съхранение на тези данни.
- в) Форма за редакция на данни за: потребители, машини, оператори, складове, обекти, задачи, типове машини и типове аварии. Във всички форми се извършва валидация на въведените данни и асинхронно обръщане към Web методи за съхранение на редактираните данни.
- д) Форми за извличане на справки – предоставя се възможност на администраторите да извличат детайлни справки. Данните в справките могат да бъдат филтрирани по различни критерии и съхранявани във excel или pdf файлове.

Приложението ще бъде разгледано в детайли в глава IV.

- *FonteFrescoMobile* – Pocket PC приложение, работещо на MS Windows Mobile. Изисква инсталиран .NET Compact Framework v2.0, желателно е и инсталирането на клавиатура за писане на кирилица. Такава клавиатура се включва в инсталационния пакет на FonteFrescoMobile. Приложението предоставя възможност за работа при наличие на интернет връзка и при липса на такава, въвеждане на информация относно извършена дейност, валидация и съхранение на въведените данни в локално в апарата или на сървъра, обновяване на необходимите данни за работата на приложението.

Приложението ще бъде разгледано в детайли в глава IV.

2.2. Бизнес слой

Реализиран е чрез Web услугата - *FonteFrescoWebService*. Работи в среда MS Windows и изисква инсталиран Web сървър с поддръжка на ASP.NET. Такъв е Microsoft IIS версия 5 или 6.

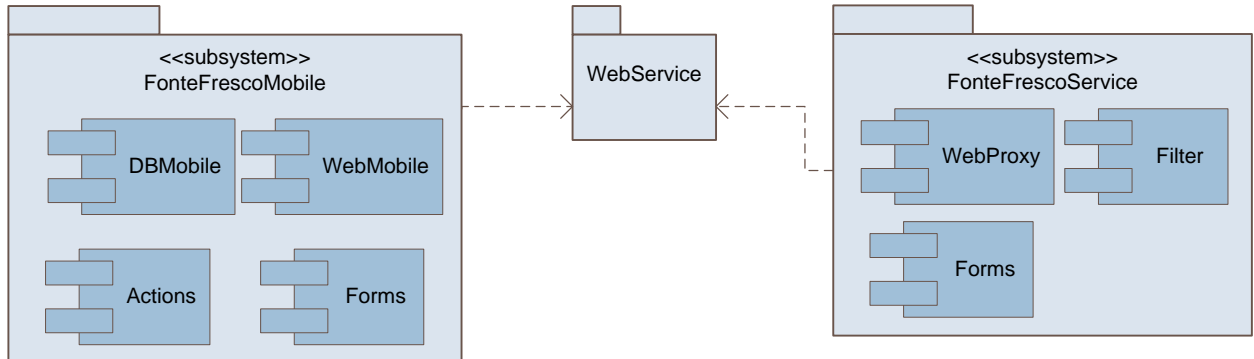
Web услугата съдържа множество методи, реализиращи определена логика като повечето от тях се обръщат към базата данни на системата. Има методи за идентификация на потребител, извличане и запис на данни, методи за построяване на различните видове справки.

2.3. Слой на базата данни.

Използваната система за управление на бази данни е MS SQL Server 2000. Базата данни е разгледана подробно в част V.

3. Модел на имплементация на системата (Implementational model)

Имплементационния модел на системата се състои от три подсистеми:



фиг. 6 Имплементационен модел на системата

Първоначалната архитектура на Pocket PC приложението, трябва да съдържа компоненти, за достъп до локално съхранените данни, за обръщение към Web услугата, форми за въвеждане на данни и обекти за съхранение на въведените данни. Администраторското приложение трябва да включва компоненти за обръщение към Web услугата, форми за въвеждане на данни, редактиране на данни и извеждане на справки и контрол за филтриране на данни.

IV Проектиране и реализация на клиентския слой

В дизайна на почти всички приложения, едно от първите решения, които се взимат е как да се имплементира графичния потребителски интерфейс, така че да предостави на потребителите интуитивен подход за работа с програмата.

При изграждането на клиентския слой на системата е използвана библиотеката на .NET Framework за изграждане на прозоречно-базиран графичен потребителски интерфейс - Windows Forms. Windows Forms предоставя възможност за бързо изграждане на потребителски интерфейс тъй като е базирана на RAD(Rapid Application Development) концепцията. RAD е подход за разработка, при който приложенията се създават визуално, чрез сглобяване на готови компоненти посредством помощници и инструменти за автоматично генериране на голяма част от кода. Windows Forms дефинира набор от класове и типове, част от които са използвани при реализирането на двете клиентски приложения на системата:

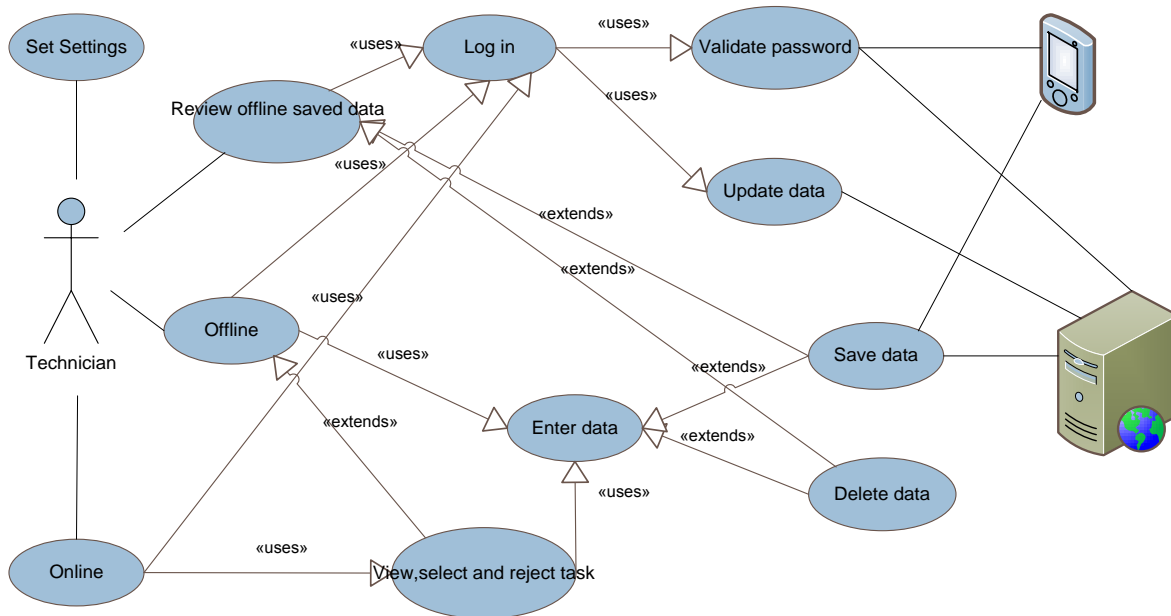
- **FonteFrescoMobile.exe** и
- **FonteFrescoService.exe.**

1. Модел на случаи на употреба- (Use Case) модел

Моделът на случаите на употреба съдържа актьорите, случаите на употреба, както и връзките между тях. Моделът на случаите на употреба се представя с помощта на UML диаграми, които показват актьорите и случаите на употреба от различни гледни точки и с различни цели.

1.1. Модел на случаите на употреба за FonteFrescoMobile приложението

Определяме един актьор за приложението – Техник, потребителят, които извършва всички взаимодействия със него.



фиг. 7 Модел на потребителските случаи – Use Case Model за FonteFrescoMobile приложението

1.1.1. Случай на употреба - Задаване на настройки (Set Settings)

Потребителят може да задава настройки на програмата: интернет адрес, време за отговор, клавиатура.

Предусловие: Потребителят се намира на първата форма от приложението „Вход в системата”.

Основен поток от действия:

1. Потребителят натиска бутон "Настройки" от първата форма на приложението- „Вход в системата”;
2. Приложението показва форма, в която потребителят може да въведе настройките за приложението;
3. Потребителят въвежда необходимата информация;

Алтернативен поток 1:

- A1 1. Потребителят потвърждава съхраняването на настройките;
- A1 2. Приложението съхранява въведените настройки;
- A1 3. Приложението показва отново формата „Вход в системата”;
- A1 4. Край.

Алтернативен поток 2:

- A2 1. Потребителят отказва съхраняването на настройките;
- A2 2. Приложението показва отново формата „Вход в системата”;
- A2 3. Край.

1.1.2. Случай на употреба - Идентифициране на потребител (Log in)

Потребителят се идентифицира в системата чрез парола и Windows CE DeviceID-то на апарата.

Предусловие: Потребителят е въвел парола в първата форма от приложението-*WelcomeForm*.

Основен поток от действия:

1. Потребителят натиска бутон "*Напред*" от първата форма на приложението- „Вход в системата”;
2. Приложението валидира въведената парола.

Алтернативен поток 1:

- A1 1. Потребителят е успешно идентифициран.
- A1 2. При наличие на интернет връзка приложението обновява локалния XML файл. В противен случай преминава в *offline* режим на работа.
- A1 3. Приложението показва на потребителя форма „Главно меню”;
- A1 4. Край.

Алтернативен поток 2:

- A2 1. Потребителят не е успешно идентифициран.
- A2 2. Приложението показва на потребителя отново форма „Вход в системата”, за да се идентифицира повторно.
- A2 3. Край.

1.1.3. Случай на употреба - Валидиране на парола (*Validate password*)

Валидиране на паролата, въведена от потребителя.

Предусловие: Потребителя е въвел парола и е натиснал бутона "*Напред*" във формата „Вход в системата”.

Основен поток от действия:

1. Изчислява се MD5 хеш стойността на въведената парола;
2. Към получения хеш се добавя *garbage*.
3. Приложението извиква web метод на web услугата за идентификация на потребител, като подава като параметри получения хеш, DeviceID-то.

Алтернативен поток 1:

- A1 1. При наличие на връзка със сървъра приложението обновява локалния XML файл, със получените данни от сървъра.
- A1 2. Приложението запазва валидната хеш стойност във файла с настройки.
- A1 3. Приложението показва на потребителя форма „Главно меню”;
- A1 4. Край.

Алтернативен поток 2:

- A2 1. При липса на връзка със сървъра приложението валидира хеш стойността спрямо тази запазена във файла с настройки(т.е. последната валидна парола) и преминава в *offline* режим на работа.
- A2 2. Приложението показва на потребителя форма „Главно меню”;
- A2 3. Край.

1.1.4. Случай на употреба - Обновяване на данни (Update data)

Данните в локалния XML файл се обновяват със данни от централната база данни, при първоначално идентифициране на потребителя или на всеки час след стартиране на приложението.

Предусловия: Потребителя се идентифицира в системата или се е задействал *timer*-а за обновяване на данни.

Основен поток от действия:

1. Приложението взема от локалните данни минималните и максимални индекси за оператори, складове, типове машини, типове аварии и задачи.
2. Приложението извиква web метод за обновяване на данни, като подава като параметри индексите.

Алтернативен поток 1:

- A1 1. При наличие на връзка със сървъра приложението получава новите данни и данните, които трябва да се изтрият(ако има такива).
- A1 2. Приложението обновява базата данни в паметта и я записва в локалния XML файл.
- A1 3. Край.

Алтернативен поток 2:

- A2 1. При липса на връзка със сървъра приложението преминава в offline режим.
- A2 2. Край.

1.1.5. Случай на употреба - Преглед на записи съхранени в апарата (Review offline saved data)

Потребителят може да преглежда записи за дейност, съхранени в апарата, по време на липса на връзка със сървъра

Предусловия: Потребителя се идентифицирал в системата, приложението е в online режим, и има съхранени записи в локалната база данни.

Основен поток от действия:

1. Приложението взема от локалните данни записа за първата дейност.
2. Потребителят преминава отново последователно по формите, в които е въвел информацията за дейността. Като във формите му се показват въведените от него данни. Потребителят може да редактира някои от данните, други не.

1.1.6. Случай на употреба - Съхранение на данни (Save data)

Потребителят може да запазва въведените от него данни за определена дейност в централната база данни или в апарата.

Предусловия: Потребителя е въвел данни или е прегледал предварително въведени данни за дейността, преминавайки последователно по формите.

Основен поток от действия:

1. Потребителят натиска бутона "Запази" на формата „Резюме“, представяща обобщена информация на въведените данни за дейността.

Алтернативен поток 1(online режим):

- A1 1. Приложението проверява дали типа на дейността, за която се отнася записът, изисква избрана задача. Ако няма не преминава на следващата точка.
- A1 2. Приложението извиква съответен Web метод на Web услугата за запазване на записът в централната база данни.
- A1 3. Край

Алтернативен поток 2(offline режим):

- A2 1. Приложението проверява ако записът е нов(т.е. няма такъв запис в локалната база данни) го добавя в локалната база данни на апарата. Ако е бил стар запис(запис, който е бил преглеждан) го обновява.
- A2 2. Приложението записва базата данни от паметта в локалния XML файл.
- A2 3. Край.

1.1.7. Случай на употреба - Изтриване на запис(Delete data)

Потребителят може да изтрива записите, които са били съхранени в апарата или такъв за който в момента е въвел данни.

Предусловия: Потребителят е на форма „Главно меню” и приложението му дава информация за броя записи съхранени в апарата и обобщена информация за първия запис; Потребителят е прегледал предварително въведени данни за дейността, преминавайки последователно по формите и се намира на формата даваща обобщена информация за дейността (форма „Резюме”); потребителят се намира на формата за проверка на номера на машина (форма „Номер на машина”) и при проверката се оказва че номера е невалиден.

Основен поток от действия:

1. Потребителят натиска бутона "Изтрий".
2. Приложението изтрива записът от локалната базата данни в паметта.
3. Край.

1.1.8. Случай на употреба - Offline работа с приложението (Offline)

Потребителят има възможност да въвежда данни и запазва информация при липса на интернет връзка.

Предусловия: Приложението е в *offline* режим на работа.

Основен поток от действия:

1. Потребителят преминава последователно по формите според типа на дейността, за която въвежда данни.
2. Приложението запазва въведените данни в локален XML файл.
3. Приложението показва формата „Вход в системата”, за да може потребителят да се идентифицира online в системата.
4. Край.

1.1.9. Случай на употреба - Online работа с приложението (Online)

Потребителят има възможност да въвежда данни и запазва информация в централна база данни при наличие на интернет връзка.

Предусловия: Потребителят е успешно идентифициран и приложението е в *online* режим на работа

Основен поток от действия:

1. Потребителят преминава последователно по формите според типа на дейността, за която въвежда данни.
2. Приложението валидира дали е избрана задача, ако типа дейност изисква такава.
3. Приложението валидира номера на машина, при въвеждане на данни за сервизни дейности и дейности за оставяне или вземане на машина от склад.
4. Приложението извиква съответен Web метод, според типа на дейността, за запазване на данните в централната база данни.
5. Приложението показва форма („Главно меню“) за да може потребителя да избере типа на следващата дейност.
6. Край.

1.1.10. Случай на употреба - Преглед, избор или отказ от задача (Review, select or reject task)

Потребителят може да преглежда задачи, да избира задача, да асоциира задача към дейност за която ще въвежда данни, или да се отказва от асоциирана задача.

Предусловия: Потребителя се намира на форма „Главно меню“.

Основен поток от действия:

1. Потребителят натиска бутона "Задачи".
2. На потребителя се показва контрол, представящ информацията за задачите му в дървовидна структура.

Алтернативен поток 1:

- A1 1. Потребителят преглежда задачи.

Алтернативен поток 2:

- A2 1. Потребителят селектира задача и натиска бутона „Потвърди“.
- A2 2. Приложението асоциира избраната задача, към дейността за която ще бъде въведена информация.
- A2 3. Край.

Алтернативен поток 3:

- A3 1. Потребителят селектира задача и натиска бутона „Отказ“.
- A3 2. Приложението премахва асоциацията между избраната задача и текущата дейност, за която се въвеждат данни.
- A3 3. Край.

1.1.11. Случай на употреба - Въвеждане на данни (Enter data)

Потребителят може да въвежда данни за определена дейност, като преминава през форми, чиято последователност зависи от указания тип на дейността. При преглед

на записи, които са били съхранени в апарата, потребителят няма възможност да редактира типа на дейността, адреса на машината(ако дейността е свързана с машина), километража.

Предусловия: Потребителят е успешно идентифицира.

Основен поток от действия:

1. Потребителят въвежда данни.

Алтернативен поток 1:

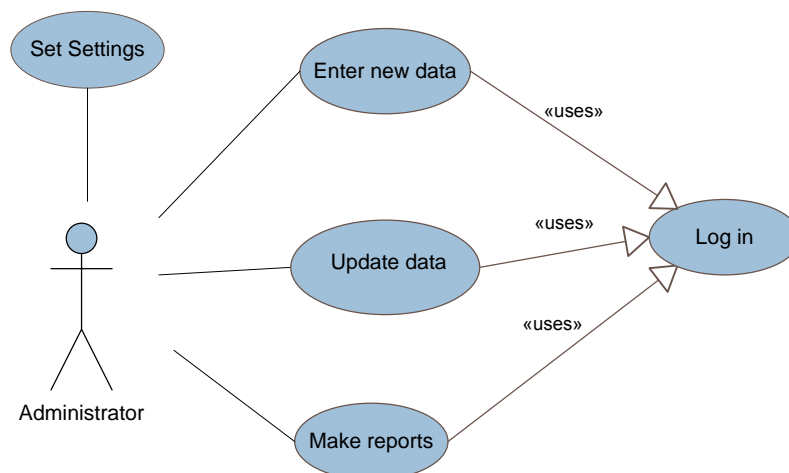
- A1 1. Потребителя натиска бутона „Напред”.
- A1 2. Въведените данни се валидират.
- A1 3. Приложението показва следващата форма на потребителя

Алтернативен поток 2:

- A2 1. Потребителя натиска бутона „Назад”.
- A2 2. Приложението показва предходната форма.
- A2 3. Потребителя редактира вече въведените от него данни.

1.2. Модел на случаите на употреба за FonteFrescoService приложението

Определяме един актьор за приложението – Администратор, потребителят, които извършва всички взаимодействия със него.



фиг. 8 Модел на потребителските случаи – Use Case Model за FonteFrescoService приложението

1.2.1. Случай на употреба - Задаване на настройки (Set Settings)

Администратора може да задава настройки на програмата: интернет адрес, време за отговор, свързване през прокси, адрес на прокси и порт на прокси.

Основен поток от действия:

1. Администратора натиска бутон "Настройки" от формата за вход в приложението или от меню "Настройки" на главната форма;
2. Приложението показва форма, в която потребителят може да въведе настройките за приложението;
3. Администратора въвежда необходимите данни;

Алтернативен поток 1:

- A1 1. Администратора потвърждава съхраняването на настройките;
- A1 2. Приложението съхранява въведената настройка;
- A1 3. Приложението показва главната форма;
- A1 4. Край.

Алтернативен поток 2:

- A2 1. Администратора отказва съхраняването на настройките;
- A2 2. Приложението показва главната форма;
- A2 3. Край.

1.2.2. Случай на употреба - Въвеждане на нови данни (Enter new data)

Администратора може да въвежда нови данни за потребители, задачи, типове аварии, типове машини, обекти, машини, оператори и складове.

Основен поток от действия:

- 1. Администратора натиска подменюто "Нов .." на съответното меню за типа данни, за който иска да въведе нов запис;
- 2. Приложението показва форма за въвеждане на нов запис за съответния тип данни;
- 3. Администратора въвежда необходимата информация;
- 4. Приложението валидира въведените данни;
- 5. Приложението извиква съответен web метод за съхранение на данните;

Алтернативен поток 1:

- A1 1. На администратора се показва съобщение че данните са съхранени успешно;
- A1 2. Приложението изтрива въведените данни от формата;
- A1 3. Край.

Алтернативен поток 2:

- A2 1. На администратора се показва съобщение, че данните не могат да бъдат съхранени;
- A2 2. Край.

1.2.3. Случай на употреба - Редактиране на данни (Update data)

Администратора може да редактира данни за потребители, задачи, типове аварии, типове машини, обекти, машини, оператори и складове.

Основен поток от действия:

- 1. Администратора натиска подменюто "Редакция на.." на съответното меню за типа данни, за който иска да направи редакция;
- 2. Приложението показва форма за редакция, съдържаща данните от съответния тип;
- 3. Администратора редактира данни;
- 4. Приложението валидира редактираните данни;
- 5. Приложението извиква съответен Web метод за съхранение на редакцията;

Алтернативен поток 1:

A1 1. На администратора се показва съобщение че данните са съхранени успешно;

A1 2. Край.

Алтернативен поток 2:

A2 1. На администратора се показва съобщение че данните не могат да бъдат съхранени;

A2 2. Край.

1.2.4. Потребителски случай извеждане на справки (Reports)

Администратора може да извежда различни детайлни справки.

Основен поток от действия:

1. Администратора натиска съответното подменю на меню "Справки", според типа справка, която желае направи;
2. Приложението извиква съответен Web метод за получаване на необходимата справка, за текущия месец;
3. Приложението показва справката на потребителя с получените данни(ако има такива);
4. Приложението зарежда филтрите;
5. Администратора указва критериите, по които иска да се филтрират данните;
6. Администратора натиска бутона "Обнови данни";
7. Приложението извиква съответен Web метод за получаване на необходимата справка, като подава като параметри стойностите зададени във филтрите;

Алтернативен поток 1:

A1 1. На администратора се показват данните според въведените от него филтри;

A1 2. Край.

Алтернативен поток 2:

A2 1. На администратора се показва съобщение че данните не могат да бъдат заредени;

A2 2. Край.

1.2.5. Случай на употреба - Идентификация на администратор (Log in)

Администратора се идентифицира в системата чрез парола. Достъп до системата се получават потребители, регистрирани като администратори.

Предусловие: Администратора е въвел парола във формата „Вход в системата”.

Основен поток от действия:

1. Администратора натиска бутона „Вход”;
2. Приложението извиква Web метод за валидиране на въведената парола;

Алтернативен поток 1:

A1 1. Администратора е успешно идентифициран;

A2 2. Приложението показва главната форма;

A1 2. Край.

Алтернативен поток 2:

A2 1. Администратора не е успешно идентифициран;

A2 2. Край.

2. Имплементация на FonteFrescoMobile

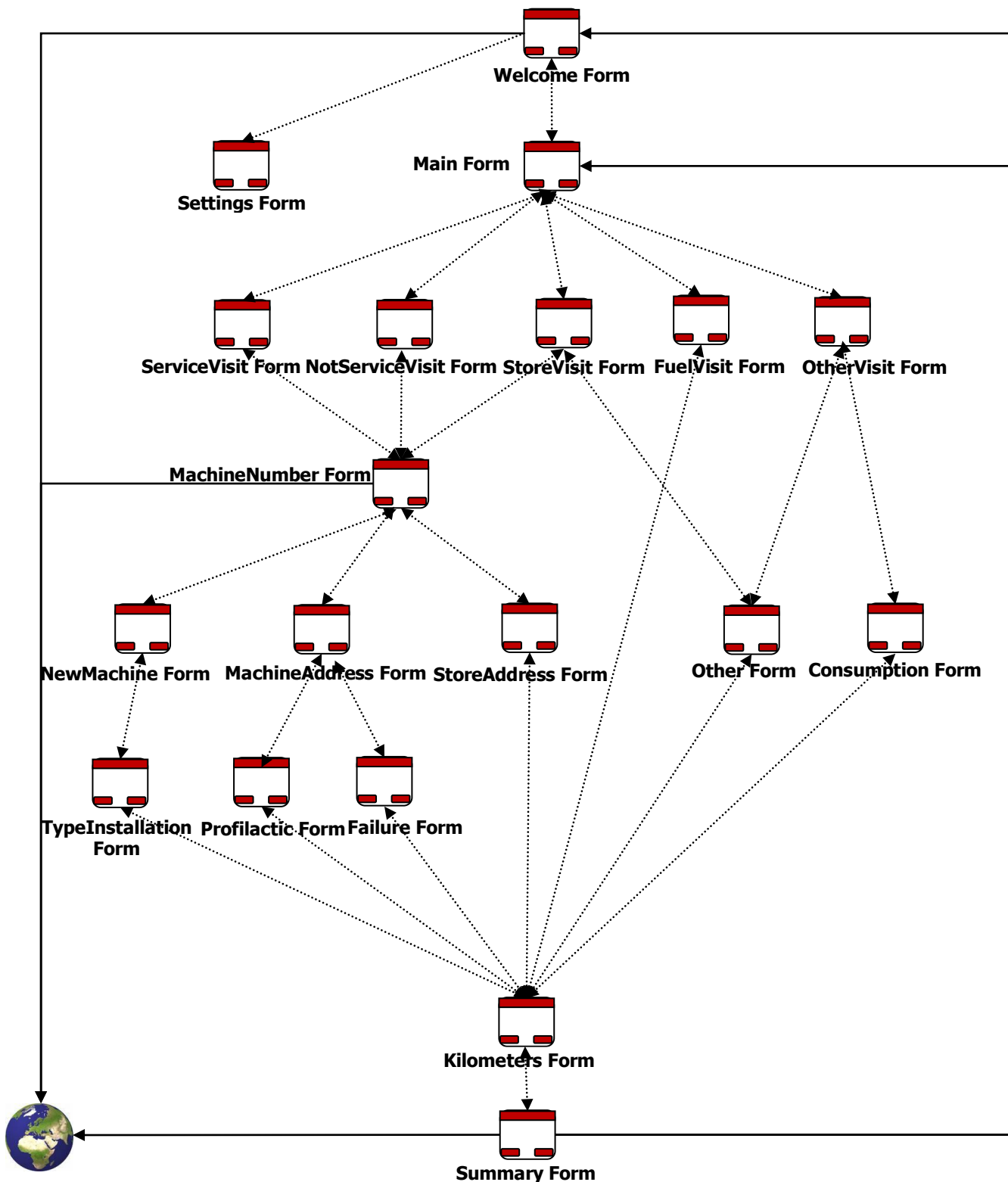
Разработката на приложения за PocketPC устройства се различава от разработката на настолни приложения. Има редица фактори, които трябва да се съобразят като памет, процесор, малки размери на екрани, батерия и др. За много от PocketPC приложенията, като и за FonteFrescoMobile, едно от основните изисквания е да бъдат лесни за употреба и да имат добро бързодействие.

За реализирането на FonteFrescoMobile приложението е използвана платформата .NET Compact Framework 2.0 и езика за програмиране C#. В глава I са представени потребителските изисквания към приложението, на базата на които е са изградени потребителските случаи на употреба на системата.

2.1. Потребителски интерфейс

Потребителския интерфейс на приложението FonteFrescoMobile е реализирано чрез последователност от форми. На потребителят се предоставя възможност да се връща назад по формите и да редактира въведената от него информация. Има няколко начина за реализация на тази функционалност. Ако реда на показване на формите на потребителя е произволен е добре да се използва стек. Последната форма в стека е последната показана форма на потребителя. Връщането назад към предишна форма, става чрез изваждане на последната форма от стека. В нашия случай последователността от форми е строго зададена, в зависимост от типа дейност, за която се въвеждат данни, или зависи от определен избор на потребителя. Затова е избран подхода в който всяка форма знае коя е следващата и пази референция към предишната форма. При достигане на последната форма, в която се представя резюме на всички въведени данни и се съхранява запис, потребителят се връща отново на основната форма за избор на тип дейност(когато е online) или на формата за вход в системата(когато е offline).

На следващата фигура е представена последователността от форми, през която се преминава в зависимост от избраната дейност във *MainForm*.



фиг. 9 Последователност от форми във FonteFrescoMobile приложението

Реализирането на приложението по този начин и въвеждането на логически свързана информация в отделни, строго последователни форми, цели да предостави на потребителя интуитивен и бърз интерфейс за работа.

2.1.1. Имплементация

Създаването на нова инстанция на форма, всеки път когато се показва, би довело до заемане на много памет от приложението. Един от начините за намаляне на използваната памет е да се преизползват обекти, особено инстанции на класа `Forms`, там където е възможно.

За тази цел всяка от формите показани на фиг. 9 е имплементирана в отделен клас, който може да има само една инстанция за приложението и предоставя глобална точка за достъп до обекта. При реализацията на класовете е използван шаблона сек (*singleton*). По този начин всички класове, за различните типове форми, имат по една инстанция в приложението и достъпът до нея се осъществява единствено чрез свойството им *instance*. В следващия код е показано как е реализиран шаблона сек за класа *ProfilacticForm*:

```
public sealed partial class ProfilacticForm : Form, IBaseForm
{
    private class Nested
    {
        internal static readonly ProfilacticForm instance = new
        ProfilacticForm();
        static Nested() { }
    }

    public static ProfilacticForm instance
    {
        get
        {

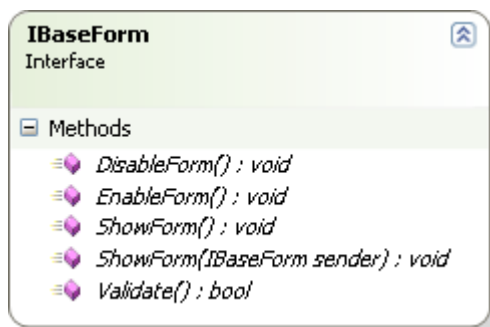
            return Nested.instance;
        }
    }
    ...
}
```

Този начин на имплементация използва `inner` клас за пълна мързелива инициализация – единствено при обръщение към *instance* ще се стартира инициализацията. По този начин инстанция на дадена форма ще се създаде едва когато е необходима(преди да се покаже).

По същия начин е имплементиран шаблона и за останалите типове форми в приложението.

Общ интерфейс за работа с формите

За да се унифицира комуникацията между формите е създаден интерфейс *IBaseForm*, който се имплементира от всички класове за формите, показани на фиг. 9.



фиг. 10 Интерфейса IBaseForm

Интерфейса декларира методи, които всеки клас за различните типове форми в приложението трябва да имплементира, за да може да се осъществи навигацията между тях по единен начин. Така показването на следваща форма или връщането на предходна не изисква информация за типа им, тъй като те се разглеждат чрез интерфейса *IBaseForm*.

Необходимите методи, които всеки клас за отделните типове форми, трябва да имплементира са:

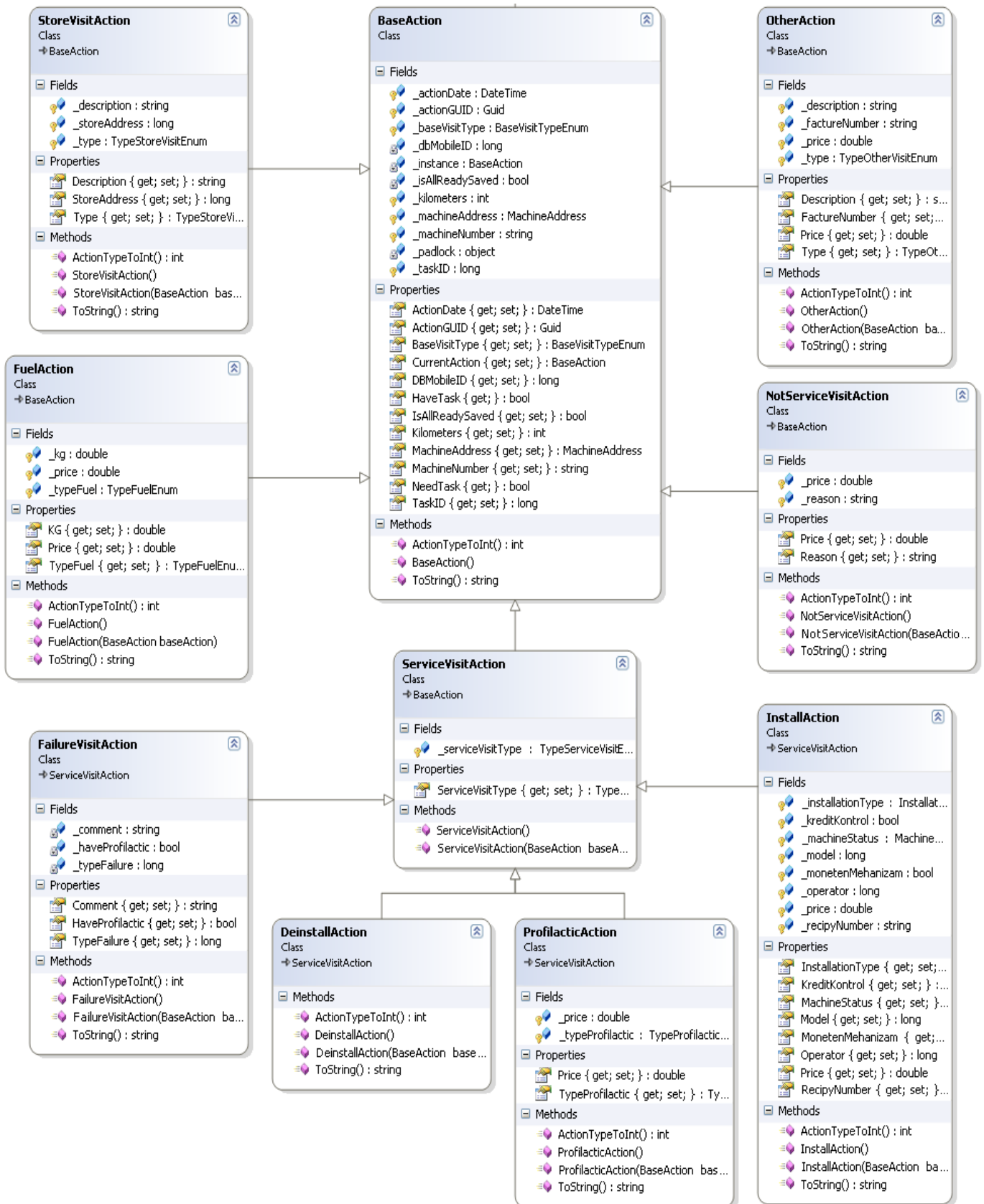
- *DisableForm()* – прави контролите във формата неспособни да приемат съобщения.
- *EnableForm()* – позволява на контролите във формата да приемат съобщения.
- *ShowForm()* – обновява данните, въведени в контролите на във формата и я показва. Този метод се вика при връщане към предишна форма, за да се обновят само данните във формата в въведени от потребителя.
- *ShowForm(IBaseForm sender)* – този метод се вика при преминаване към следваща форма. Подадения параметър *sender* е референция към предходната форма. Метода извиква *ShowForm()*.
- *Validate()* – валидира въведените от потребителя данни.

Методите *DisableForm* и *EnableForm* се използват от контрола *WaitTimerControl*, който се показва при обръщение към Web сървъра. Тъй като това става автоматично на всеки час, контрола може да бъде показан върху произволна форма и затова той разглежда своя родителски контрол чрез интерфейса му *IBaseForm*. По този начин *WaitTimerControl* деактивира и след като приключи обръщението отново да активира формата, върху която е показан, без да се взема под внимание нейният конкретен тип.

Запазване на въведените от потребителя данни

За да се реализира изискването за редакция на вече въведени данни при връщане назад по формите е необходимо тези данни да се съхраняват. За тази цел е създадена йерархия от класове (*Actions*). Свойствата на всеки клас, отговарят на данните, които са необходими за съхранение на запис за определена дейност. Общите свойства на всички класове са изнесени в базовия клас *BaseAction*. Всеки от наследниците му добавя свойства, специфични за отделния тип дейност. Например при дейност от тип Зареждане на гориво трябва да се въведат и съхранят данни за тип, цена и кг/л на

зареденото гориво, като се присвоят съответно на свойствата на обект от тип *FuelAction*: *KG*, *Price* и *TypeFuel*.



фиг. 11 Клас диаграма на класовете от пространството за имена *CompactClient.Actions*

Във всеки един момент въведените данни в приложението се присвояват само на един обект *Action*, който е текущ и той се достъпва чрез свойството на *BaseAction* - *CurrentAction*. Тъй като свойството е от тип *BaseAction* то може да съдържа референция към всеки един от наследниците на *BaseAction*.

При преминаване през различните форми, всяка форма разглежда свойството *CurrentAction* като обект от съответния тип *Action*, чийто свойства може да асоциира с контролите си. Това става във методите:

- *GetFormStateFromAction* – свойствата на съответния тип *Action* се показват във контролите на формата.
- *CopyFormStateToAction* – въведените данни се присвояват на свойствата на обекта от съответния тип *Action*.

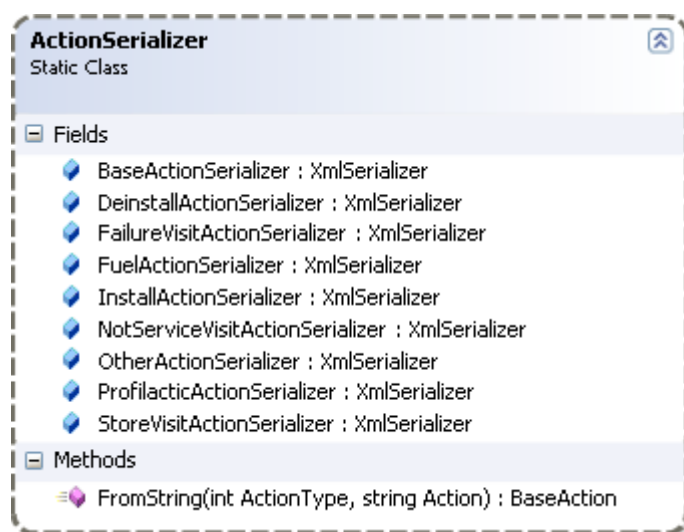
Тези методи се извикват съответно преди да се покаже формата и преди да се скрие формата, и да се премине на следваща.

Свойството *CurrentAction* има следните атрибути

- *XMLIgnore* – стойността на свойството няма да се сериализира при *Serialize* метода на *XMLSerializer* класа.
- *FormatObjectIgnore* – *custom* атрибут, със свойство *Inherited* = *false*, т.е. *CurrentAction* свойството няма да се наследи от наследниците на *BaseAction*.

В класа *BaseAction* се дефинира виртуален метод *ActionTypeToInt*, който връща *int* стойност от изброения тип *BaseVisitTypeEnum*, отговаряща на *_baseVisitType* член променливата(т.е. отговаряща на базовия тип на дейността). Във всеки наследник се предефинира този метод, така че да се връща съответната стойност, отговаряща на типа на класа.

Другия метод на *BaseAction* е предефинирания метод *ToStirng*. Метода връща обекта сериализиран към XML документ. За сериализацията и десериализацията на *Actions* обектите се използва статичния клас *ActionSerializer*.



фиг. 12 Action Serializer класа

Класа предоставя статични инстанции на *XMLSerializer* класа, за всеки тип *Action*. Тези инстанции сериализират обекти от конкретния тип в XML документ. Всеки от наследниците на *BaseAction* предефинира метода *ToString* използвайки съответния *XMLSerializer* инстанцииран чрез неговия тип.

Статичния метод *FromString* се използва за десериализиране на *Actions* обекти, по подадена стойност за типа на обекта и сериализирания обект. Стойността за типа на обекта съответства на стойността на типа в изброяения тип *BaseActionEnum*. Десериализирането на обекти се използва преди преглед на записи, съхранени в апарата по време на *offline* режима на работа на програмата.

2.2. Реализация на offline режима

Приложението *FonteFrescoMobile* трябва да предоставя на потребителя възможност за работа и при липса на интернет връзка. Въведените от потребителя данни трябва да се съхраняват в апарата и при поява на интернет връзка да се съхранят на сървъра. В глава II са представени различните стратегии за съхранение на данни при *PocketPC* приложения: релационна база данни(*Microsoft SQL Server CE*), в локален файл (XML файл), и базирана на сесии структура данни, в паметта.

В приложението *FonteFrescoMobile* данните се съхраняват в локален XML файл, който се зарежда в паметта при стартиране на приложението. Избора за използване на XML файл за съхранение на данни се ръководи от няколко факта:

- Количеството данни, което трябва да се съхранява на *PocketPC* в повечето случаи ще бъде между 50 и 100 kb. Зареждането им в паметта ще има по-добра производителност от колкото достъп до тях през *SQL Server CE Query Engine*.
- Обмена на данни между *PocketPC* и *Web Service* ще е малък(около 1Mb).
- Употребата на XML и *Web Service* е предпочитана, и *.NET Compact Framework* предоставя широка поддръжка и на двете технологии.

Локалния XML файл има следната схема:

```
<xs:schema id="DBMobile" targetNamespace="http://www.changeme.now/DBMobile.xsd"
xmlns:mstns="http://www.changeme.now/DBMobile.xsd"
xmlns="http://www.changeme.now/DBMobile.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" attributeFormDefault="qualified" elementFormDefault="qualified">
  <xs:element name="DBMobile" msdata:IsDataSet="true">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="TypeMachine">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ID" type="xs:long" />
              <xs:element name="Name">
                <xs:simpleType>
                  <xs:restriction base="xs:string">
                    <xs:maxLength value="50" />
                  </xs:restriction>
                </xs:simpleType>
              </xs:element>
              <xs:element name="InsertUpdateIndex" type="xs:long" default="0" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:choice>
    </xs:element>
  </xs:schema>
```

```

<xs:element name="Operators">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:long" />
      <xs:element name="Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="100" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="InsertUpdateIndex" type="xs:long" default="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="TypeFailure">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:long" />
      <xs:element name="Name">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="255" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="InsertUpdateIndex" type="xs:long" default="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="StoreAddress">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:long" />
      <xs:element name="OperatorID" type="xs:long" />
      <xs:element name="StoreCity">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="StoreAddress">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="255" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="InsertUpdateIndex" type="xs:long" default="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="MinIndexes">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" msdata:AutoIncrement="true" type="xs:long" />
      <xs:element name="MinOperators" type="xs:long" default="0" />
      <xs:element name="MinTypeMachine" type="xs:long" default="0" />
      <xs:element name="MinTypeFailure" type="xs:long" default="0" />
      <xs:element name="MinStoreAddress" type="xs:long" default="0" />
      <xs:element name="MinTasks" type="xs:long" default="0" />
    </xs:sequence>
  </xs:complexType>
</xs:element>

```

```

<xs:element name="Actions">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" msdata:AutoIncrement="true" type="xs:long" />
      <xs:element name="ActionType" type="xs:int" />
      <xs:element name="SerializedAction">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="4000" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Tasks">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="ID" type="xs:long" />
      <xs:element name="ObektID" type="xs:long" />
      <xs:element name="ObektName" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="100" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="ObektAddress">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="255" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="ObektCity">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="50" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="TypeActivityAsNumber" type="xs:long" />
      <xs:element name="InsertUpdateIndex" type="xs:long" default="0" />
      <xs:element name="Comment" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="100" />
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
<xs:unique name="Constraint1" msdata:PrimaryKey="true">
  <xs:selector xpath="./mstns:TypeMachine" />
  <xs:field xpath="mstns:ID" />
</xs:unique>
<xs:unique name="Operators_Constraint1" msdata:ConstraintName="Constraint1"
  msdata:PrimaryKey="true">
  <xs:selector xpath="./mstns:Operators" />
  <xs:field xpath="mstns:ID" />
</xs:unique>
<xs:unique name="TypeFailure_Constraint1" msdata:ConstraintName="Constraint1"
  msdata:PrimaryKey="true">
  <xs:selector xpath="./mstns:TypeFailure" />
  <xs:field xpath="mstns:ID" />
</xs:unique>

```

```

<xs:unique name="StoreAddress_Constraint1"
  msdata:ConstraintName="Constraint1" msdata:PrimaryKey="true">
  <xs:selector xpath="//mstns:StoreAddress" />
  <xs:field xpath="mstns:ID" />
</xs:unique>
<xs:unique name="MinIndexes_Constraint1" msdata:ConstraintName="Constraint1">
  <xs:selector xpath="//mstns:MinIndexes" />
  <xs:field xpath="mstns:ID" />
</xs:unique>
<xs:unique name="Actions_Constraint1" msdata:ConstraintName="Constraint1"
  msdata:PrimaryKey="true">
  <xs:selector xpath="//mstns:Actions" />
  <xs:field xpath="mstns:ID" />
</xs:unique>
<xs:unique name="Tasks_Constraint1" msdata:ConstraintName="Constraint1"
  msdata:PrimaryKey="true">
  <xs:selector xpath="//mstns:Tasks" />
  <xs:field xpath="mstns:ID" />
</xs:unique>
</xs:element>
</xs:schema>

```

Във схемата на XML файла е дефинирана последователност от елементи за всеки тип данни, за които трябва да се съхранява информация: типове машини, оператори, типове аварии, складове на оператори, дейности, минимални индекси и задачи.

Данните за типове машини, оператори, типове аварии, складове и задачи се получават от централната база данни. За да се намали интернет трафика, при обновяването на тези данни се изпращат само най-големите стойности за *InsertUpdateIndex* елементите за всеки тип данни (типове машини, оператори, типове аварии, складове, задачи) и стойността им за съответния елемент за минимален индекс в *MinIndex*. Като резултат на потребителят се връщат само новите данни или данните, които трябва да се обновят (т.е тези със стойност за *InsertUpdateIndex* по-голяма от подадената) и данните които трябва да изтрият (т.е тези със стойност за *InsertUpdateIndex* по-малко от подадената отрицателна стойност за съответния тип). Преди да се изтрие даден елемент, с неговия *InsertUpdateIndex* се обновява минималния индекс за типа данни, към който принадлежи в *MinIndex*. Така се запазва индекса на последния елемент, който е изтрит.

В елементите *Actions* се запазват данните за дейностите, които са били въведени по време на *offline* режима на работа на програмата. За всяка дейност се съхранява *ID* (autoincrement), *ActionType* – int стойността за типа дейност изброяения тип *BaseVisitTypeEnum* и *SerializedAction* – сериализирания обект. *ActionType* е необходимо да се съхранява за да може да се десериализира обекта.

В елемента *MinIndex* се съхранява индексите на последно изтритите записи за всеки от типовете данни: типове машини, оператори, типове аварии, складове и задачи.

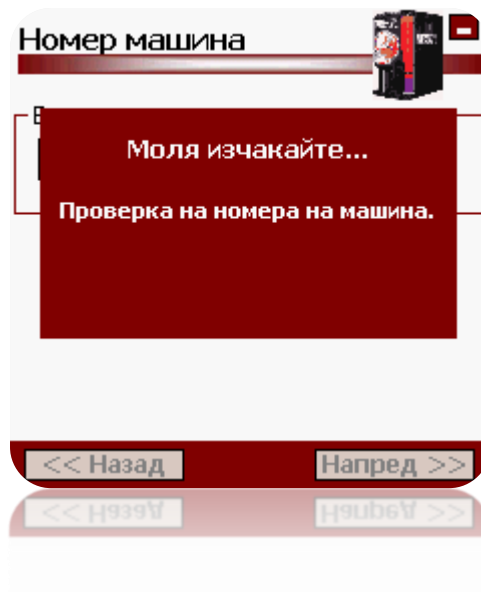
Методите за работа със локалния XML файл са дефинирани в класа *DBMobile*. Той е типизиран *DataSet*, наследник на *System.Data.DataSet* класа и е генериран от горната XML схема в *FonteFrescoWebService* приложението. .NET позволява разделяне на дефиницията на класове, структури или интерфейси във повече от един файл, използвайки ключовата дума *partial*. По този начин във *FonteFrescoMobile* е разширена функционалността на класа *DBMobile* като са добавени методи, които са необходими във *PocketPC* приложението:

- Метод за зареждане на данните от локалния XML файл - *LoadDataBase*.
- Метод за записване на данните в локалния XML файл - *SaveDataBaseToFile*.
- Метод за проверка дали има данни в *DBMobile* - *IsDBMobileEmpty*.
- Методи за обновяване на данните за типове машини, оператори, типове аварии, складове и задачи от подаден обект от тип *DBMobile* – *UpdateOperatorsTable*, *UpdateTasksTable*, *UpdateTypeMachineTable*, *UpdateTypeFailureTable*, *UpdateStoreAddressTable*.
- Методи за извличане на най-големите стойности за *InsertUpdateIndex* елементите за всеки тип данни – *GetSmallestTasksIndex*, *GetBiggestTasksIndex*, *GetSmallestStoreAddressIndex*, *GetBiggestStoreAddressIndex* и т.н.
- Метод за изтриване на запис за задача по подадено *ID* на задача - *DeleteRecordForTask*.
- Метод за изтриване на запис за дейност по подадено *ID* на дейност - *DeleteRecordForAction*.
- Метод за обновяване на дейност при подаден обект от базовия тип *BaseAction* - *UpdateDataBase*.
- Метод за добавяне на дейност при подаден обект от базовия тип *BaseAction*. – *InsertAction* и др.
- Свойство *CountActionsForInssert* за броя записи за дейности съхранени в апарата.
- Свойство *CountTasks* за броя на задачите.
- Свойство *FirstAction* за получаване на първата дейност съхранена в локалната база данни.

Достъпът до методите се осъществява през статичното свойство на класа *DBMobile* *instance*, което инициализира инстанция на *DBMobile*, ако няма такава създадена.

2.3. Реализация на online режима

FonteFrescoMobile приложението обменя данни със сървъра чрез асинхронно извикване на Web услуга. При асинхронното извикване на Web услуга потребителския интерфейс не замръзва докато приключи обръщението. Във FonteFrescoMobile при извикване на Web услугата на потребителят се показва следния *WaitTimerControl*, със текст съответстващ на извършваната операция:



фиг. 13 Извикване на Web услуга

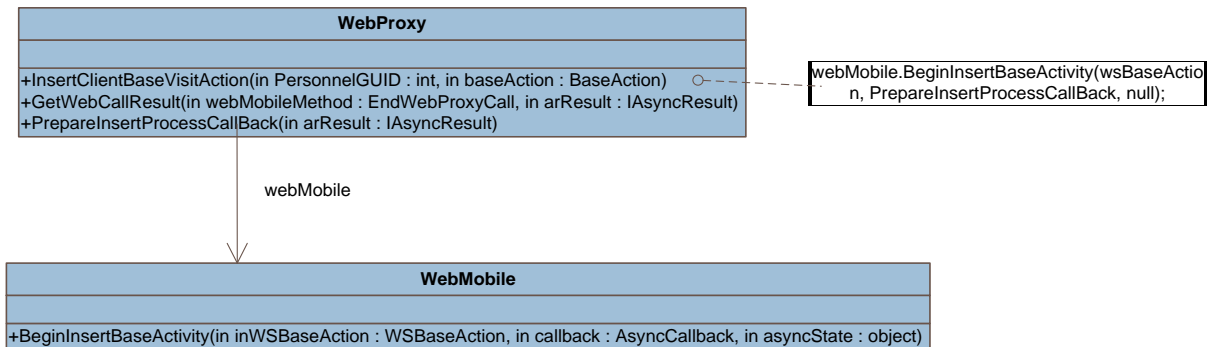
Извикването на Web услугата се контролира от класа `WebProxy`, реализиран чрез шаблоните пълномощно(proxy) и сек(singleton). Шаблонна сек се използва за да се осигури само един обект от този клас и той да се създаде тогава когато наистина се наложи да се използва. Шаблонна е имплементиран отново чрез `inner` клас, за да се осигури пълна мързелива инициализация:

```
public class WebProxy
{
    private class Nested
    {
        internal static readonly WebProxy instance = new WebProxy();
        static Nested()
        {
        }
    }
    ...

    public static WebProxy instance
    {
        get
        {
            return Nested.instance;
        }
    }
    ...
}
```

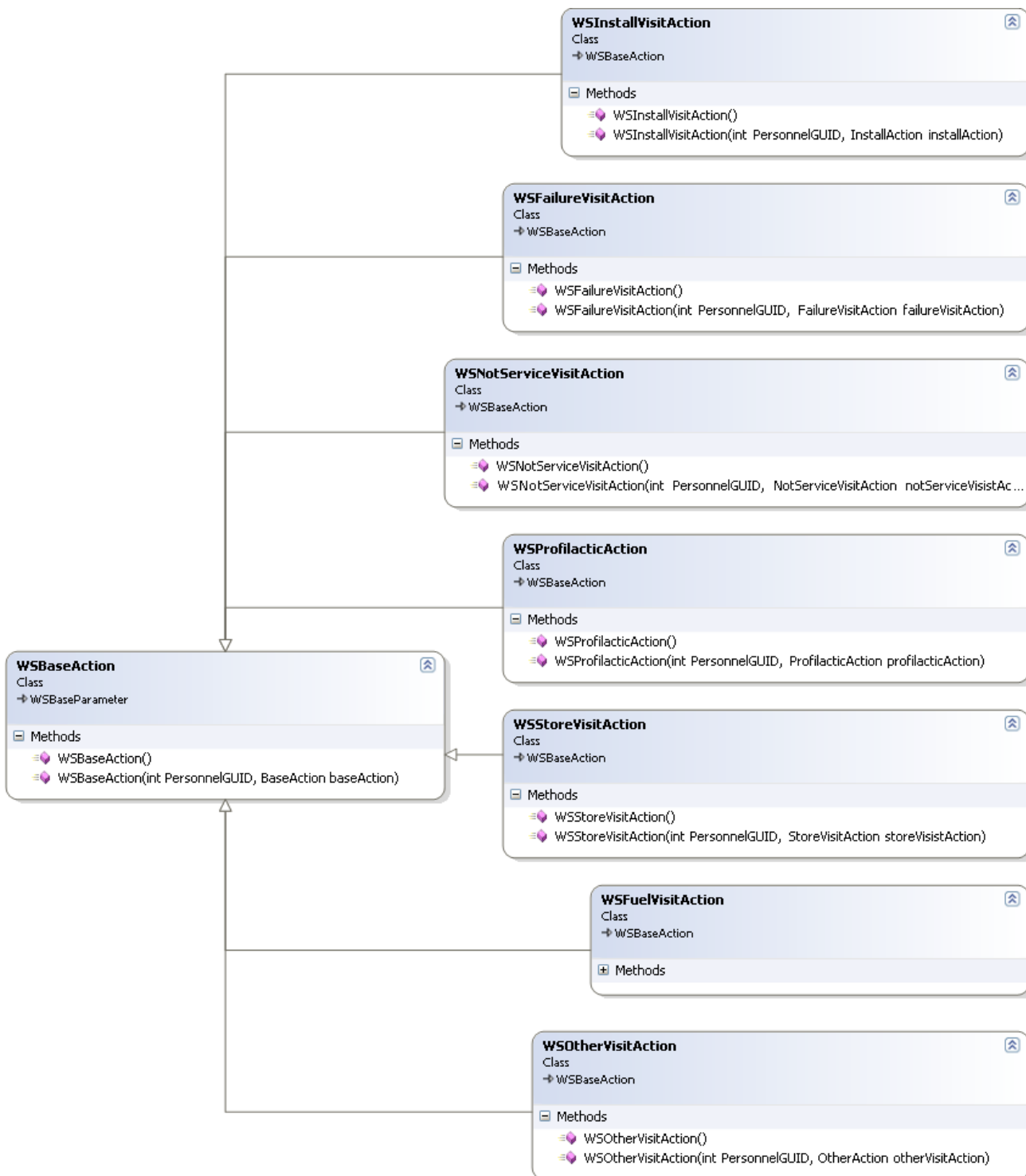
`WebProxy` класа представлява отдалечено пълномощно (*remote proxy*), разновидност на шаблона пълномощно, който предоставя локален представител на обект от друго адресно пространство. Този вид пълномощно се среща в литературата под името „посредник“. `WebProxy` класа съдържа референция към, автоматично генерирания клас, за достъп до Web услугата. В `WebProxy` класа са дефинирани методи, съответстващи на методите предоставени от Web услугата. В класа се централизира

обработката на общи изключения, като изключения възникнали при обмен на данни със сървъра или изключения при автентикация на потребителя. Това става в метода *GetWebCallResult*.



фиг. 14 Класове за достъп до web услуги, имплементиращи шаблона отдалечено пълномощно

На фиг. 15 са представени обекти (*WSAction*), които се изискват от Web услугата и са дефинирани в нея. Всеки от тях дефинира свойства, необходими за добавянето на запис за определена дейност. Във *FonteFrescoMobile* приложението дефиницията на тези класове е разширена в *partial* клас, като за всеки от тях са добавени два конструктора: един подразбиращ се, и един приемащ GUID-то на потребителя за сесията и съответен обект от *Action* обектите, които се използват в приложението за съхранение на данните, въведени за определена дейност. В конструктора на всеки от *WSAction* - ите се прави копиране на необходимите свойства от подадения *Action* обект.



фиг. 15 Клас диаграма на класовете от пространството на имена *CompactClient.WebActions*

За да се намали интернет трафика, данните, изпращани към сървъра се компресират. За тази цел е използвана open - source библиотеката за компресиране *SharpZipLib* за .NET Framework.

За да може да се маркира всеки Web метод с атрибут за компресия, се създава клас *CompressionSoapExtensionAttribute*, наследник на класа *SoapExtensionAttribute*:

```

using System;
...
using ICSharpCode.SharpZipLib.GZip;

namespace WSCompress
{
    [AttributeUsage(AttributeTargets.Method)]
    public class CompressionSoapExtensionAttribute : SoapExtensionAttribute
    {
        private int priority;
        public override Type ExtensionType
        {
            get { return typeof(CompressionSoapExtension); }
        }

        public override int Priority
        {
            get { return priority; }
            set { priority = value; }
        }
    }
}

```

Свойството *ExtensionType* връща типа имплементиращ допълнителната логика (*CompressionSoapExtension*), а свойството *Priority* указва реда на изпълнение, когато има няколко допълнения едновременно.

Имплементацията на конкретната логика за добавяне на функционалността е клас, който е наследник на *SoapExtension* (в *System.Web.Services.Protocols* пространството от имена) :

```

public class CompressionSoapExtension : SoapExtension
{
    Stream oldStream;
    Stream newStream;

    public override Stream ChainStream(Stream stream)
    {
        oldStream = stream;
        newStream = new MemoryStream();
        return newStream;
    }

    public override object GetInitializer(LogicalMethodInfo methodInfo,
    SoapExtensionAttribute attribute)
    {
        return attribute;
    }

    public override object GetInitializer(Type type)
    {
        return typeof(CompressionSoapExtension);
    }

    public override void Initialize(object initializer)
    {
        CompressionSoapExtensionAttribute attribute =
            (CompressionSoapExtensionAttribute)initializer;
    }
}

```

```

public override void ProcessMessage(SoapMessage message)
{
    switch (message.Stage)
    {
        case SoapMessageStage.AfterSerialize: GZipCompress();break;
        case SoapMessageStage.BeforeDeserialize: GZipDecompress();break;
    }
}

private void GZipCompress()
{
    Byte[] buffer = new Byte[2048];
    int size = 2048;
    newStream.Seek(0, SeekOrigin.Begin);
    GZipOutputStream zipOutputStream = new GZipOutputStream(oldStream);
    while (true)
    {
        size = newStream.Read(buffer, 0, buffer.Length);
        if (size > 0)
            zipOutputStream.Write(buffer, 0, size);
        else
            break;
    }
    zipOutputStream.Flush();
    zipOutputStream.Close();
}

private void GZipDecompress()
{
    Byte[] buffer = new Byte[2048];
    int size;
    GZipInputStream zipInputStream = new GZipInputStream(oldStream);
    size = 2048;
    while (true)
    {
        size = zipInputStream.Read(buffer, 0, buffer.Length);
        if (size > 0)
        {
            newStream.Write(buffer, 0, size);
        }
        else
            break;
    }
}

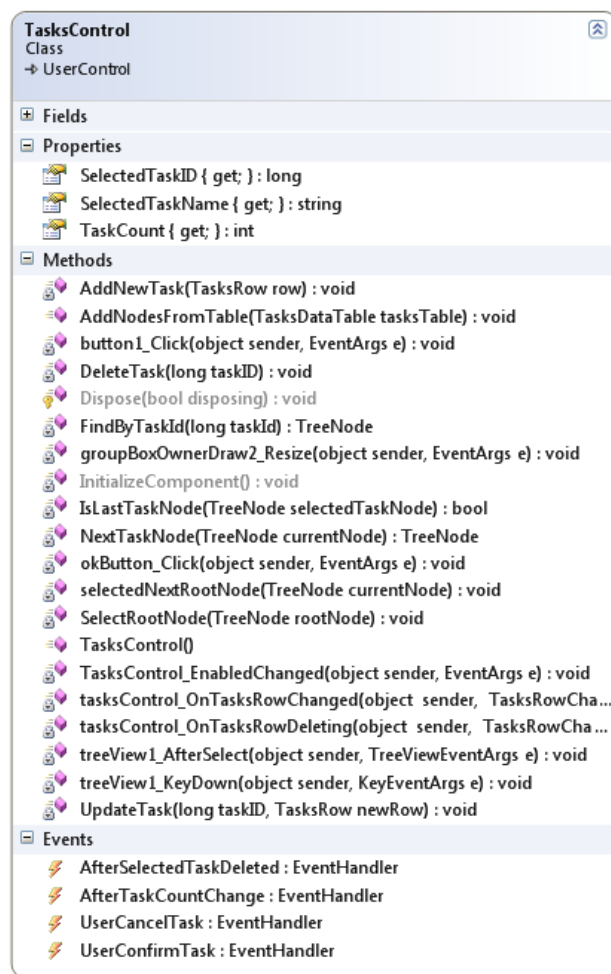
```

Основния метод, *ProcessMessage*, се извиква на всяка фаза от обработката на SOAP съобщението. Тук имат значение само *AfterSerialize* и *BeforeDeserialize* фазите. *AfterSerialize* фазата указва, че съобщението е сериализирано и тук трябва да се компресира. *BeforeDeserialize* фазата указва, че съобщението е пристигнало и е готово да се десериализира, и тук трябва не сериализираните все още данни да се декомпресират. По този начин всеки Web метод може да се маркира с атрибута за компресия [[CompressionSoapExtension](#)] и в дефиницията на Web услугата и при клиента в *Reference.cs* файла.

Hyper Text Transfer Protocol (Http) е транспортния протокол, който използва Web услугата.

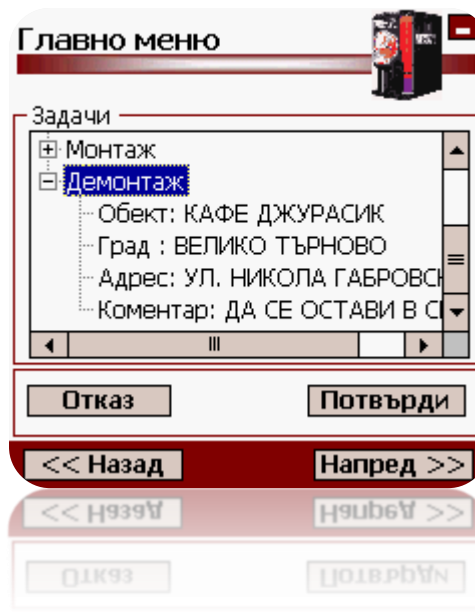
2.4. User контроли

За да може да се преизползва на няколко места функционалността за показване и избиране на задачи, е имплементиран *TasksControl* класа, наследник на *UserControl*. Контрола предоставя възможност за зареждане на задачите от *Tasks* таблицата на типизирания dataset *DBMobile*, във *TreeView* контрол (дървовидна колекция от именувани обекти, всеки представен чрез *TreeNode*). Контрола предоставя методи за добавяне на задача в дървото по подаден ред - *TasksRow* от таблицата *Tasks*, обновяване на задача, премахване на задача от дървото. Чрез свойствата на контрола могат да се извлече информация за броя задачи, името на избраната задача и ID-то на задачата. *TasksControl*-а дефинира и събития, които известяват обектите абонирани за тях, за изтриване на избрана задача, смяна на броя на задачи, за отказ от задача или за избор на задача.



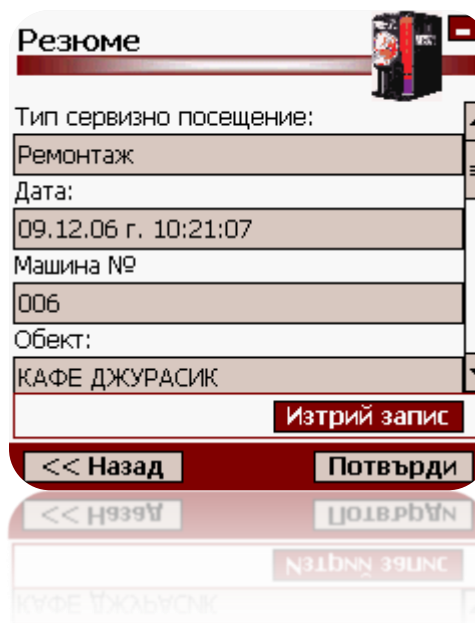
фиг. 16 Класа *TasksControl*

В *MainForm* контрола със задачи изглежда по следния начин:



фиг. 17 Контрол за преглед на задачи

Класовете от пространството на имена *SummaryUserControls* дават възможност за показване на всички данни, въведени от потребителя за определена дейност. За всеки тип дейност е създаден отделен контрол наследник на *UserControl*, а всички класове имплементират интерфейса *ISummaryUserControl*, деклариращ общите свойства и методи, които трябва да се имплементират в наследниците. В метода *Init* на всеки от наследниците се подава като параметър определен *Action* обект, чийто свойства се показват от контрола:



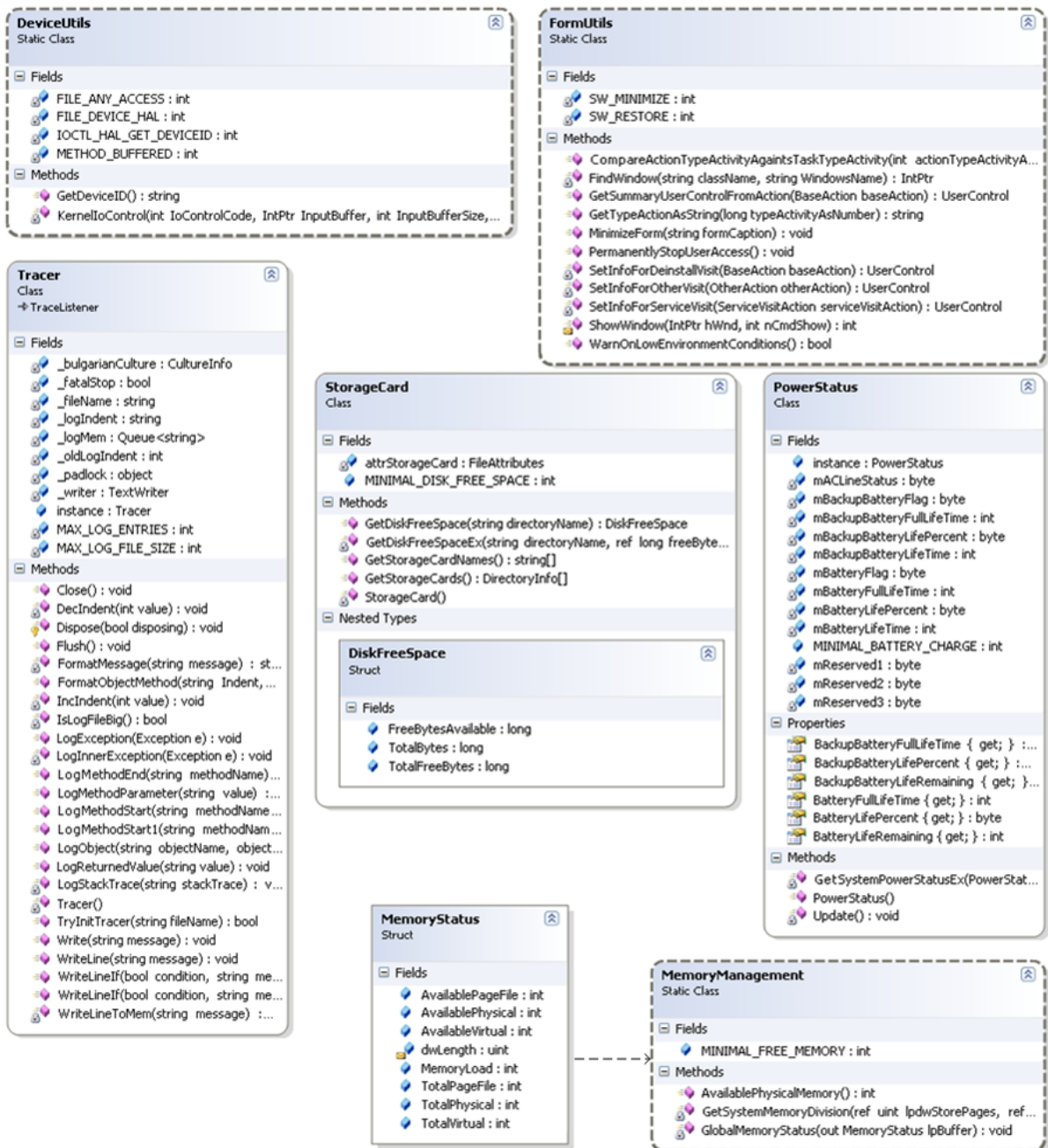
фиг. 18 Форма „Резюме” с *InstallUserControl*

Имплементацията на класовете може да се види в кода предоставен към дипломната работа.

2.5. Utils класове

Класовете в пространството от имена *CompactFrameworkUtils* са:

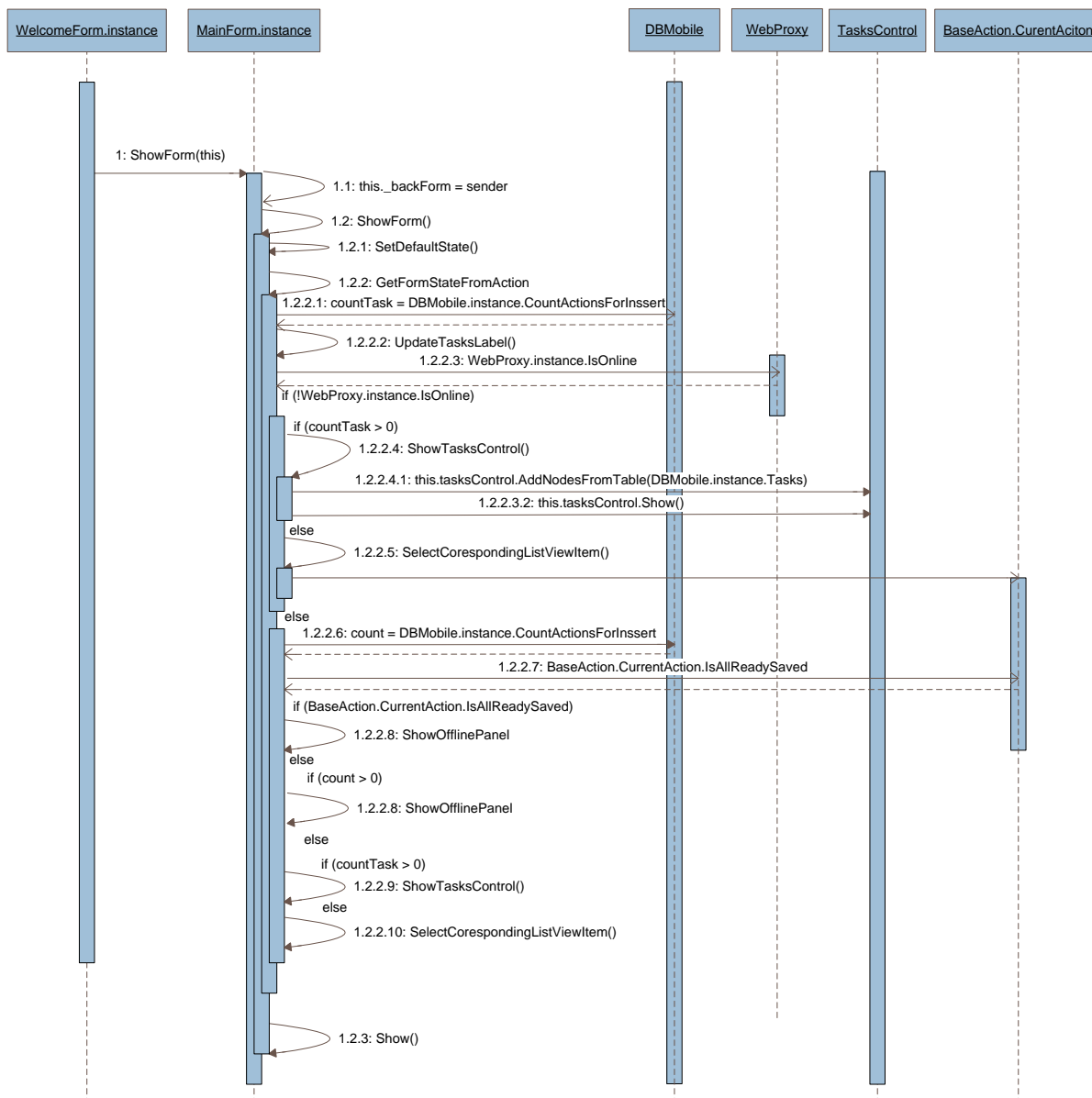
- *DeviceUtils* – статичен клас дефиниращ метод за извличане на Windows CE DeviceID-то. Няма *managed* поддръжка в .NET Compact Framework за извличане на DeviceID-то на апарата. Вместо това трябва да се използва Platform Invocation Services(P/Invoke) за извикване на unmanaged Win32 API функцията *KernelIoControl*.
- *FormUtils* – статичен клас дефиниращ методи с глобално предназначение:
 - *WarnOnLowEnvironmentConditions* – проверява последователно батерията, дисковото пространство и наличната памет. Този метод се извиква преди преминаване към следваща форма, преди обръщение към Web услугата и преди съхранение на данни в локалната база данни.
 - *GetSummaryUserControlFromAction* – връща инстанция на клас от контролите в *SummaryUsersControls* пространството от имена , съответстващ на подадения параметър и попълнен със свойствата му.
 - *GetTypeActionAsString* – връща локализиран низ съответстващ на подадената int стойност на типа дейност.
 - *CompareActionTypeActivityAgainstTaskTypeActivity* – връща true или false в зависимост дали типа на дейността съвпада с типа на дейността зададена в задачата.
 - *MinimizeForm* – извиква се за минимизиране на формата, чийто *caption* е подаден като параметър. Извикват се Win32 API *FindWindow* и *ShowWindow*.
- *MemoryManagement* – статичен клас, предоставящ метод за получаване на наличната памет.
- *PowerStatus* – предоставя свойства за извличане на информация за оставащите проценти на батерията, на допълнителната батерия, оставащото време в секунди и др.
- *StorageCard* - предоставя достъп до подвижната storage карта на устройството.
- *TextWriterTraceListener* – предоставя възможност за logging на изключения, обекти, параметри и др.



фиг. 19 Клас диаграма на класовете в пространството от имена CompactClient.CompactFrameworkUtils

2.6. Динамична реализация

След като са разгледани основните класове, в следващите диаграми на последователности ще бъдат показани какви обекти се създават и как си взаимодействат в различни потребителски случаи.



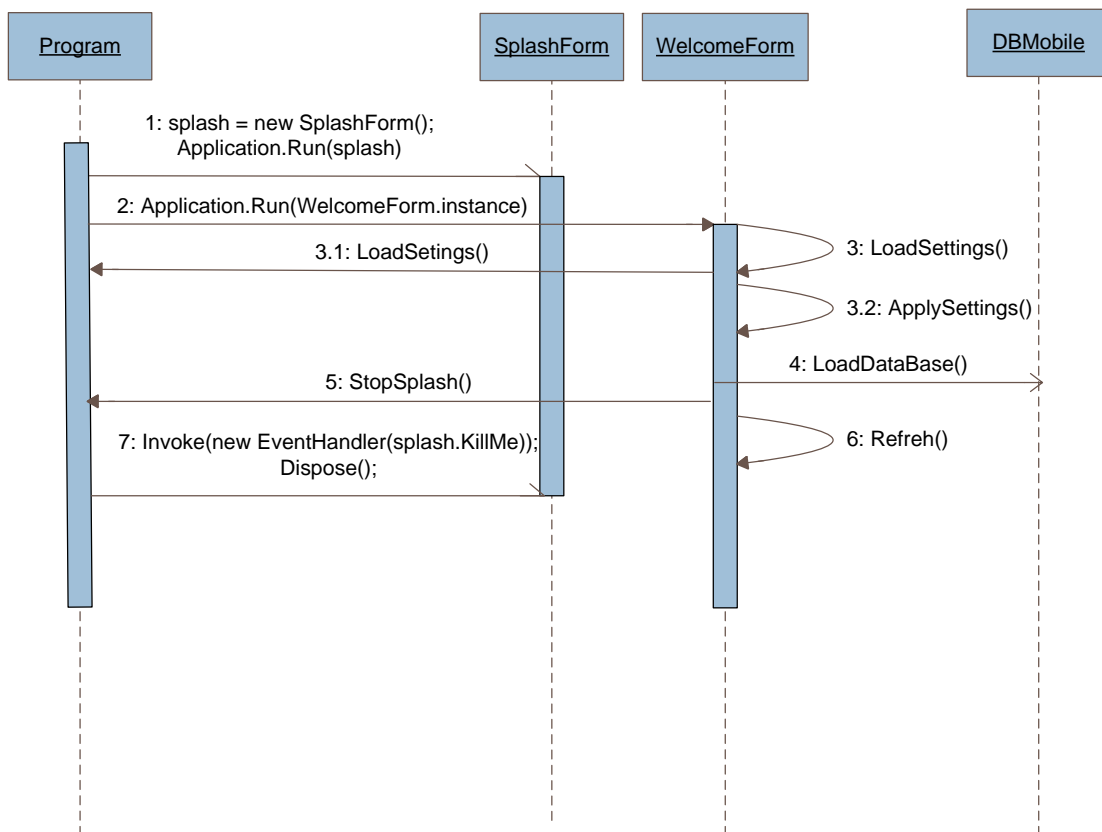
фиг. 20 Диаграма на последователността за метода ShowForm(IBaseForm sender) на класа MainForm

На диаграмата е представена базовата последователност от съобщения, които се изпращат между обектите, при показването на форма „Главно меню“. В зависимост от наличието на интернет връзка, записи съхранени в апарата и задачи, формата може да се покаже по различен начин:

- При наличие на интернет връзка и ако има записи, съхранени в апарата по-време на *offline* режима на работа на програмата, се показва панел предоставящ на потребителя информация за броя записи, съхранени в телефона и обобщена информация за първия запис.
- В противен случай се проверява дали има получени задачи. Ако има такива на потребителя се дава възможност да избере задача от контрола за задачи.

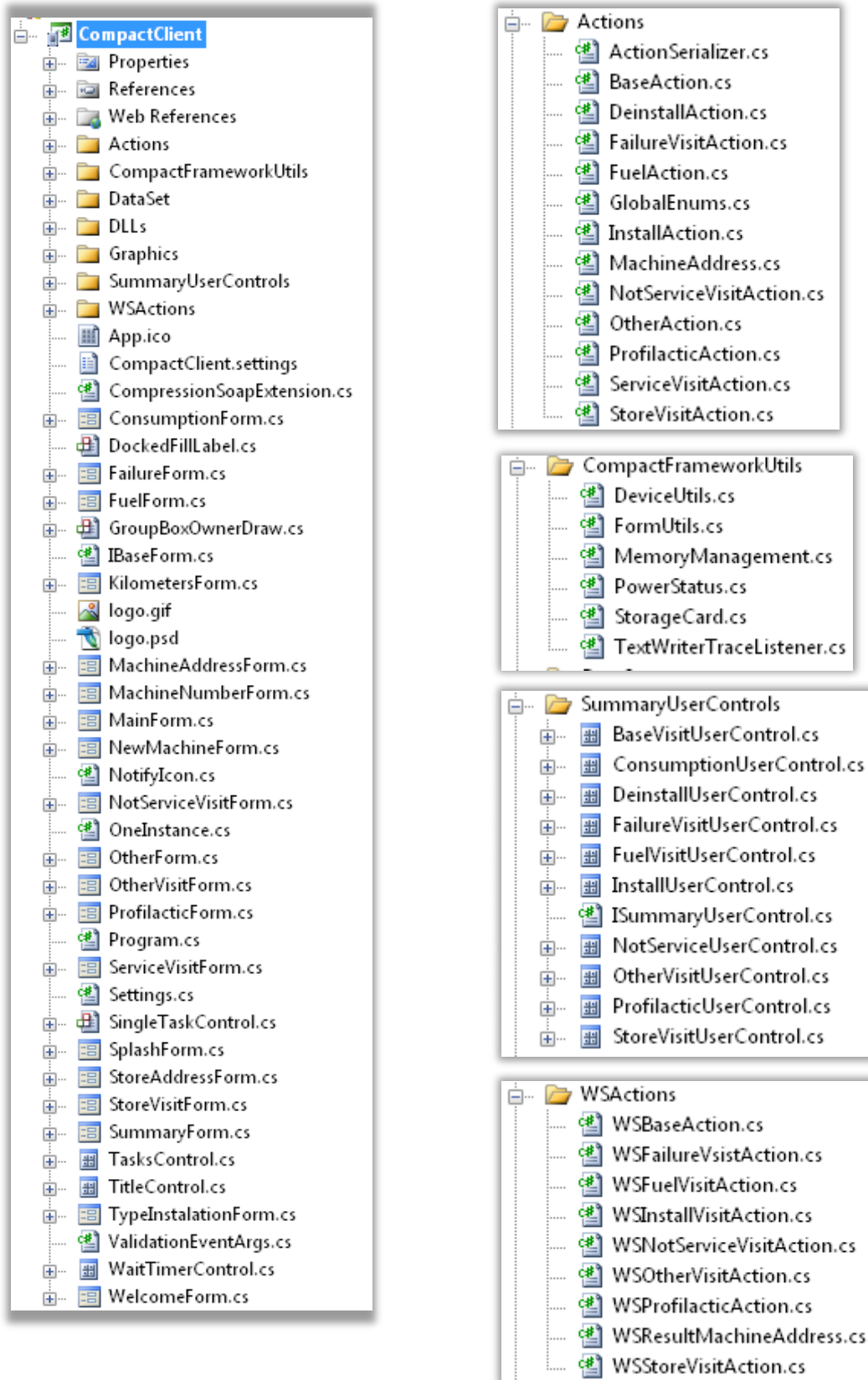
- Ако няма нито съхранени записи, които да се преглеждат, нито задачи на потребителя се показва списък с дейности, в който трябва да укаже типа дейност за която желае да въвежда данни.

На следващата диаграма е представен процеса на стартиране на приложението. При стартиране на потребителя се показва форма (*SplashForm*), която се стартира на отделена нишка, докато се инициализира и зареди формата „Вход в системата”. След приключване на инициализацията и зареждането на настройките, и локалния XML файл в паметта, се извиква функция за скриване на *Splash* формата.



фиг. 21 Диаграма на последователностите при стартиране на приложението

2.7. Структура на приложението



фиг. 22 Структура на приложението

3. Имплементация на FonteFrescoService

За реализирането на FonteFrescoService приложението е използвана платформата .NET Framework 2.0 и езика за програмиране C#.

В глава I са представени потребителските изисквания към приложението.

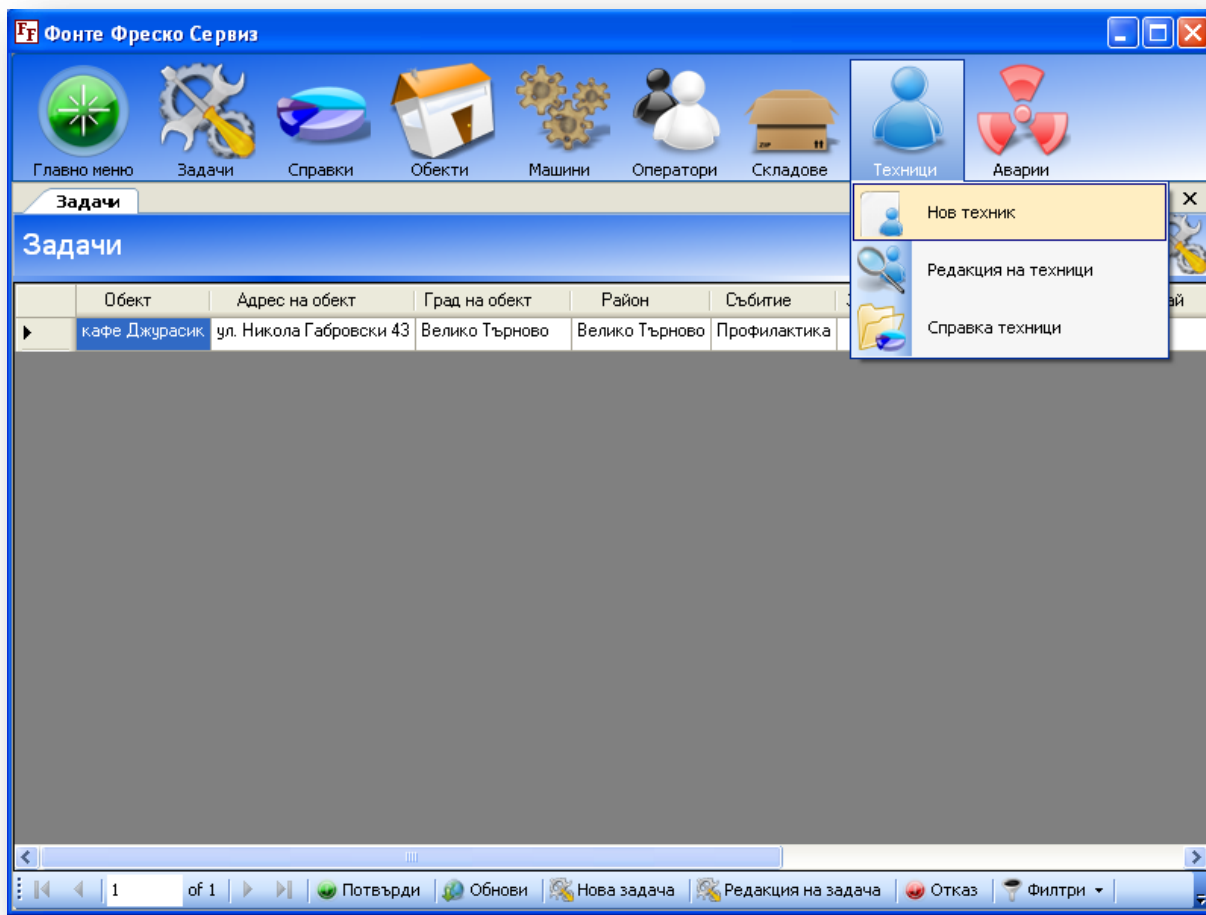
3.1. Потребителски интерфейс

3.1.1. MDI(Multiple Document Interface) приложение

MDI приложенията позволяват на потребителите да работят с различен брой документи(или форми) в едно и също време, като всеки документ се показва в различен прозорец. Всички прозорци са *child* прозорци за един базов – *parent* прозорец. Указването на MDI *parent* става чрез присвояване на стойност true за свойството *IsMDIContainer*, на формата, която е избрана за главна.

В приложението е използван *DockManager* контрол на *Weifen Luo*, който реализира *docking* възможностите за MDI форми. Той предоставя свойства като: MDI tabbed интерфейси, автоматично скриване на форми, свободно drag-and-drop и др. Компонента се използва, като всяка *child* форма наследява *DockContainer* класа, който се намира в пространството от имена *WeifenLuo.WinFormsUI*.

Форма „*Фонте Фреско Сервиз*” е главната форма на приложението, в която се отварят всички останали форми, избрани от администратора.



фиг. 23 Главна форма на приложението „Фонте Фреско Сервиз“

На администратора е предоставен бърз и удобен начин за работа чрез групираните менюта:

- Задачи
- Справки
- Обекти
- Машини
- Оператори
- Складове
- Техници
- Аварии

Селектирайки дадено меню, от подменюто администратора може да укаже вида на операцията която ще извършва:

- въвеждане на нови данни от съответния тип
- редакция на съществуващи данни
- извличане на справка за данните от този тип

Във всички MDI child форми съобщенията за грешки при обръщение към сървъра се показват в *TitleControl*-а на формата. Контрола е наследник на класа *UserControl* и предоставя свойства за заглавие на формата, графика, и текст за грешка.

3.1.2. Форми за въвеждане на данни

На администратора се предоставя възможност за въвеждане на нови данни, от определен тип във отделна форма. Формите за въвеждане на данни са:

- **Форма „Нов техник“;**
- **Форма „Нов тип авария“;**
- **Форма „Нов склад“;**
- **Форма „Нов оператор“;**
- **Форма „Нова машина“;**
- **Форма „Нов тип машина“;**
- **Форма „Нов обект“;**
- **Форма „Нова задача“;**

Интерфейсът и функционалността на формите е близка, затова в точката ще бъде разгледана само формата „Нов техник“. Останалите форми са представени в приложението към дипломната работа.

The screenshot shows a web application window titled "Фонте Фреско Сервиз". The main menu bar includes icons for "Главно меню", "Задачи", "Справки", "Обекти", "Машини", "Оператори", "Складове", "Техници", and "Аварии". The current page is "Нов техник". The form fields are as follows:

- Администратор:
- Име:
- Парола:
- Повтори парола: (with a red error icon and message "Двете пароли не съвпадат")
- Личен номер:
- Телефон:
- Адрес:
- Град:
- Активен:

Buttons at the bottom: "Потвърди" (green) and "Отказ" (red).

фиг. 24 Валидация на въведената парола във Форма „Нов потребител“

Формата предоставя възможност на администратора да въведе: име, парола, личен номер, телефон, адрес, град и да укаже типа на потребителя и дали е активен. За валидацията на въведените данни във формите за добавяне на нов запис е използван

third party *Validator* контрол. Този контрол предоставя възможност за валидиране на контроли от потребителския интерфейс, с изключително малко писане на код. Валидацията на контролите се указва в прозореца със свойства (*property window*) на контрола. Указва се типа на валидация (*Require, Regular Expression* и др.), режима (*Focus Change, Submit* или и двете), свойства на *ErrorProvider*-а и др. Свойствата *RequiredMessage on validator, RegularExpressionMessage on validator, CompareMessage on validator, RangeMessage on validator*, позволяват да се укаже съобщение, което да се покаже на потребителя при неуспех на съответната валидация.

При съхранението на данни се прави асинхронно извикване на съответния web метод. Прихващането на изключения става в *callback* метода, като при изключение се показва съобщение с възникналата грешка, в *TitleControl*-а на формата.

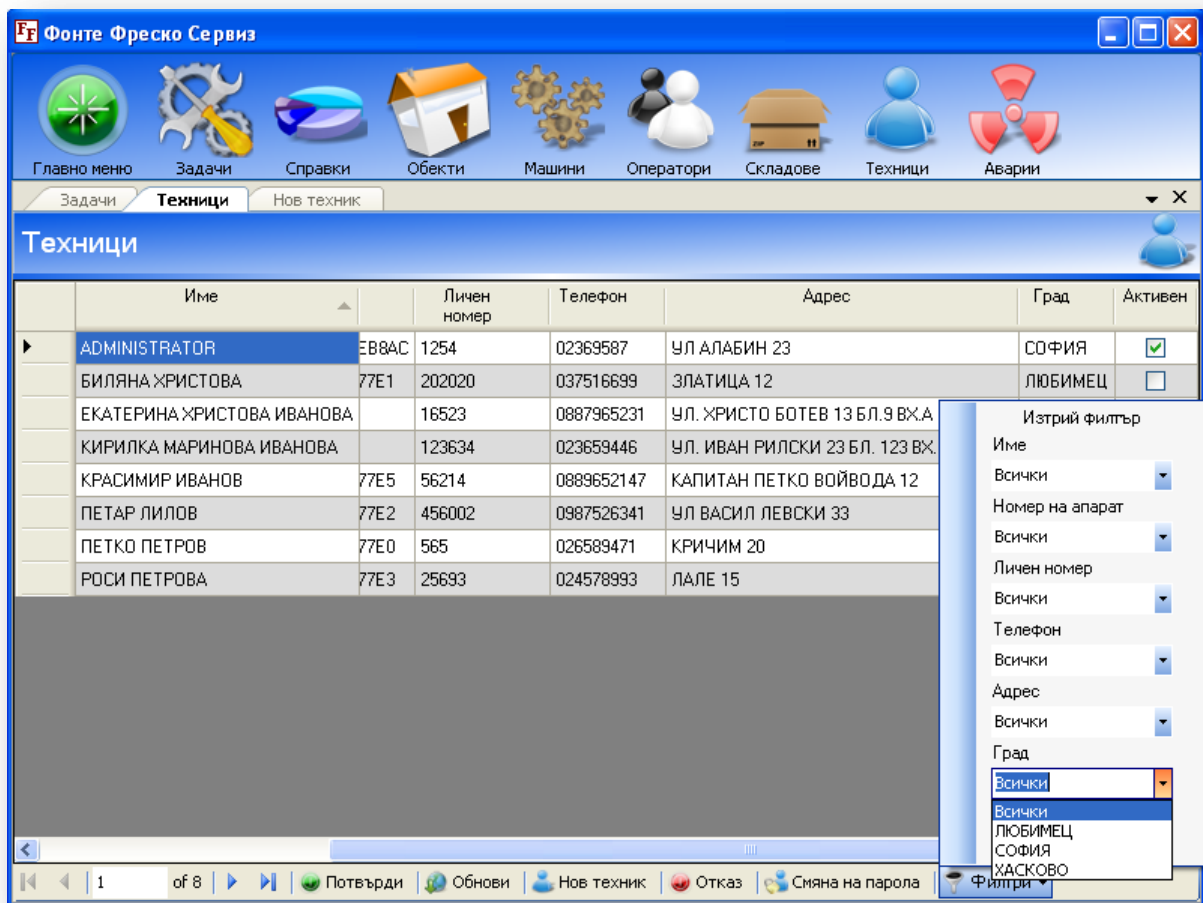
Имплементацията на формата е представена в приложението към дипломната работа.

3.1.3. Форми за редакция на данни

Редакцията на данни се извършва във една от следните форми според типа на данните:

- **Форма „Техници“;**
- **Форма „Типове аварии“;**
- **Форма „Складове“;**
- **Форма „Оператори“;**
- **Форма „Машини“;**
- **Форма „Типове машини“;**
- **Форма „Обекти“;**
- **Форма „Задача“;**

Интерфейсът и функционалността на формите е близка, затова в точката ще бъде разгледана само формата „Техници“. Останалите форми са представени в приложението към дипломната работа.



фиг. 25 Форма „Техници”

За представянето на данните са използвани следните контроли:

- *DataGridView* – контрола, предоставя възможност за представяне и редактиране на данни в табличен вид от различни типове data sources. *DataGridView* контрола е свързан към *BindingSource* компонент, който е свързан към типизирания *DataSet – UsersDataSet*. За данните, които не могат да бъдат редактирани е присвоена стойност true на свойството на колоната *ReadOnly*. Добавянето на нов ред в таблицата е забранено. Добавянето на запис става в отделна форма, която се показва на администратора от бутона „Нов техник”.
- *BindingNavigator* – контрола предоставя стандартизиран начин за търсене и обхождане на данни. Той също е свързан към *BindingSource* компонента, към който е свързан и *DataGridView* контрола. По този начин предвижвайки се чрез *BindingNavigator* контрола, в колекцията от данни на *BindingSource*, се променя съответно и селектирания ред в *DataGridView* контрола. Към стандартните бутони на *BindingNavigator*-а са добавени допълнителни бутони за запазване на направените промени („Потвърди”), за обновяване на данните в таблицата („Обнови”),

за добавяне нов техник („Нов техник”), за премахване на направените промени („Отказ”) и др.

- *FilterToolStripDropDownButton* – за да се изпълни изискването за филтриране на данни в таблицата, е създаден custom контрол *FilterToolStripDropDownButton*. Контрола е наследник на *ToolStripDropDownButton* контрола и предоставя възможност за указване на стойности, по които да се филтрират данните в колони от *DataGridViewColumnCollection*. Свързването на контрола към колони, върху които трябва да се извърши филтриране става чрез метода му
BoundToColumns(DataGridViewColumnCollection columns, BindingSource filteredSource)

като за параметри се подават колекцията от колони на таблицата и *BindingSource*, към който е асоциирана таблицата и върху който се изпълнява филтъра Колоните, по които се филтрира могат да бъдат от един от следните типове: *FilteredDataGridViewCheckBoxColumn*, *FilteredDataGridViewTextboxColumn*, *FilteredDataGridViewLinkColumn*, *FilteredDataGridViewImageColumn*, *FilteredDataGridViewComboBoxColumn*. Те са наследници на съответните типове *DataGridView* колони, и в конструктора си указват *DefaultHeaderCellType* свойството на колоната да бъде типа на класа *DataGridViewFilteredColumnHeaderCell*. Това е клас, който е наследник на *DataGridViewColumnHeaderCell*, и предефинира виртуалния му метод *GetFormattedValue*.

При запазването на направените промени се прави асинхронно извикване на съответния Web метод. Прихващането на изключения става в *callback* метода, като при изключения възникнали при комуникацията със сървъра (*WebException* или *SoapException*) се показва съобщение с възникналата грешка, в *TitleControl*-а на формата. Грешки възникнали при обновяването на базата данни се прихващат в Web услугата, за да се локализира и се добавят на свойството *RowError* на реда предизвикал изключение. Администратора вижда иконата и съобщението за грешка в началото на реда в таблицата.

Имплементацията на формата е представена в приложението към дипломната работа.

3.1.4. Форми за справки

Формите за справки предоставят възможност на администратора да извлича детайлни справки. При реализирането на справките е използван *Visual Studio ReportViewer* контрол. Той поддържа режим на локална обработка и позволява да се стартират клиентски report definition(.rdlc) файлове, използвайки вградените възможности за обработка на контрола. RDLC файлът представлява XML файл, описващ дизайна на *report*-а.



фиг. 26 Форма „Справка разходи”

Дизайн на справки

При дизайна на справките са използвани регионите за данни: таблица, матрица и списък. Всички справки съдържат *footer* със системната дата, броя страници, текущата страница и имената на администратора. Името на администратора се запазват в свойството на класа *Program - UserName*, след успешната му идентификация. За да се визуализира в справката името на администратора се присвоява на параметър *UserName*, добавен във всяка от справките.

В лявата част на някои от формите за справки има списък - карта на документа. Картата на документа представя в дървовиден вид групите, по които са групирани данните в справката. При избиране на някоя група автоматично се отива на страницата с данните от съответната група. Това е особено полезно при работа със много данни.

Локализация на справки

ReportViewer контрола предоставя *ToolStrip* контрол със бутони за предвижване по страниците на справката, за принтиране, за преглед в оформление за принтиране, за експортиране на справката към excel или pdf файл и за търсене. За локализиране на съобщенията на *ReportViewer* контрола е имплементиран класа *ReportViewerLocalizedMessages*, който имплементира интерфейса *IreportViewerMessage* и предефинира свойствата му. Локализирането на съобщенията става като се присвои на свойството *Messages* на *ReportViewer* контрола инстанция на класа *ReportViewerLocalizedMessages*.

Филтриране на справки

Филтрите на справките се указват в падащи списъци под справката. Падащите списъци са свързани чрез *BindingSource* към типизирани *DataSet*-и. При показването на формата се извиква Web метод, който връща като резултат *DataSet* за филтрите за съответната справка и *DataSet* с данните за текущия месец за справката. След като е указан филтъра администратора трябва да натисне бутона „Обнови данни” за да получи филтрирания резултат.

При обновяването на данни и филтри се прави асинхронно извикване на съответния Web метод. Прихващането на изключения става в *callback* метода, като при изключение се показва съобщение с възникналата грешка, в *TitleContorl*-а на формата.

Администратора има възможност да извършва различни видове справки в следните форми:

- Форма „Справка техници”
- Форма „Справка типове аварии”
- Форма „Справка складове”
- Форма „Справка оператори”
- Форма „Справка машини”
- Форма „Справка типове машини”
- Форма „Справка обекти”
- Форма „Справка приключени задачи”
- Форма „Справка Nestle”
- Форма „Справка профилактики – обща”
- Форма „Справка профилактики – подробна”
- Форма „Справка нови монтажи – обща”
- Форма „Справка нови монтажи – подробна”
- Форма „Справка сервизни дейности – по оператори”
- Форма „Справка сервизни дейности – по техници”
- Форма „Справка пътен лист”
- Форма „Справка техници – обща”
- Форма „Справка техници – подробна”
- Форма „Справка монтажи и ремонтажи”
- Форма „Справка история на машина”

- Форма „Справка време за реакция”
- Форма „Справка приходи”
- Форма „Справка разходи”
- Форма „Справка последен монтаж на машина”

Имплементацията на формата за справка е представена в приложението към дипломната работа.

3.2. Извикване на Web услуга

FonteFrescoService приложението обменя данни със сървъра чрез асинхронно извикване на Web услуга. При асинхронното извикване на Web услуга потребителския интерфейс не замръзва докато приключи обръщението. Във FonteFrescoService при извикване на Web услугата на потребителят се показва следния *WaitControl*, custom контрол, докато обръщението приключи. При всяко обръщение се извършва автентикация на потребителя., чрез custom SOAP header атрибута. За да може да се маркира всеки Web метод с custom атрибут за автентикация на потребител, е създаден клас *PostLogOnAuthenticationSoapExtensionAttribute*, наследник на класа *SoapExtensionAttribute*:

```
[AttributeUsage(AttributeTargets.Method)]
public class PostLogOnAuthenticationSoapExtensionAttribute :
SoapExtensionAttribute
{
    private int priority;

    public override Type ExtensionType
    {
        get { return typeof(PostLogOnAuthenticationSoapExtension); }
    }

    public override int Priority
    {
        get { return priority; }
        set { priority = value; }
    }
}
```

Свойството *ExtensionType* връща типа имплементиращ допълнителната логика (*PostLogOnAuthenticationSoapExtensionAttribute*), а свойството *Priority* указва реда на изпълнение, когато има няколко допълнения едновременно.

Имплементацията на конкретната логика за добавяне на функционалността е клас, който е наследник на *SoapExtension* (в *System.Web.Services.Protocols* пространството от имена) :

```

public class PostLogOnAuthenticationSoapExtension : SoapExtension
{
    public override object GetInitializer(LogicalMethodInfo methodInfo,
                                         SoapExtensionAttribute attribute)
    {
        return attribute;
    }

    public override object GetInitializer(Type type)
    {
        return typeof(PostLogOnAuthenticationSoapExtension);
    }

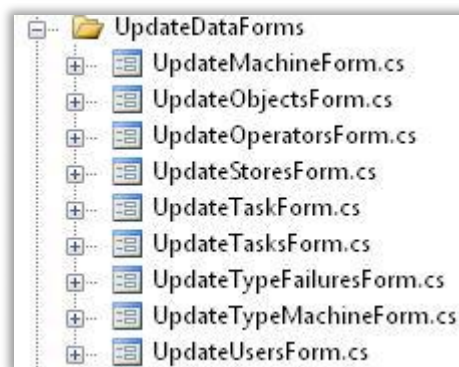
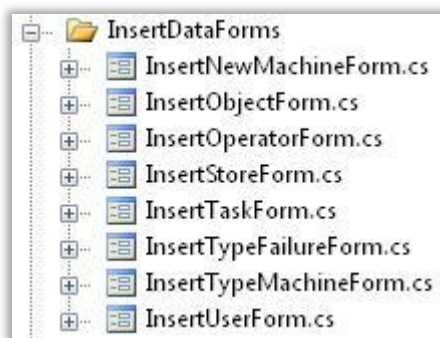
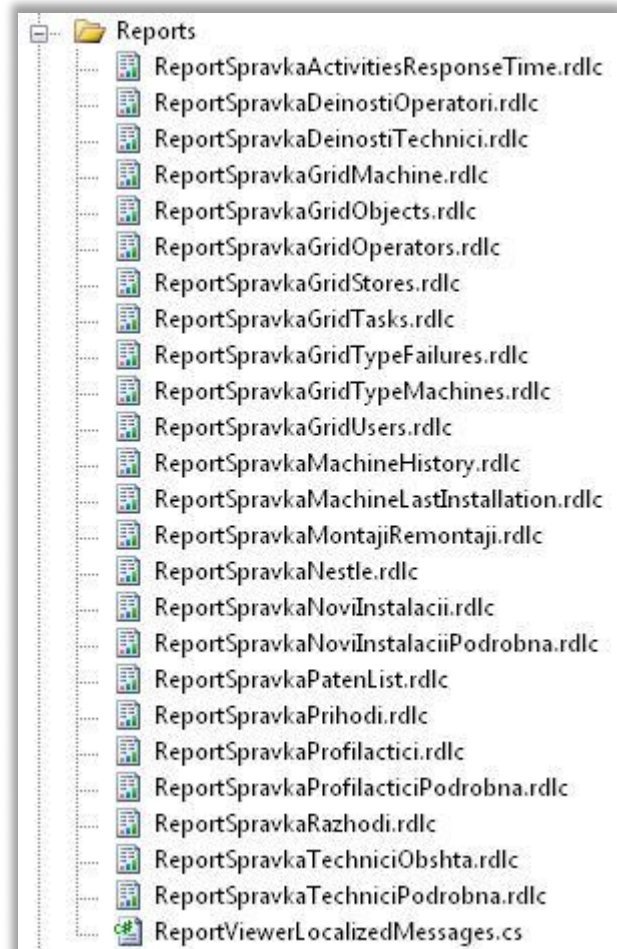
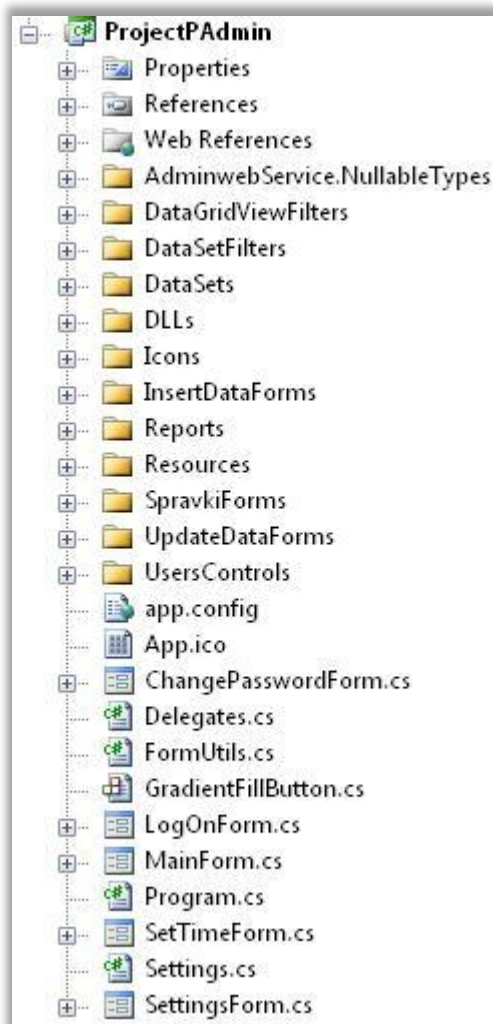
    public override void Initialize(object initializer)
    {
        PostLogOnAuthenticationSoapExtensionAttribute attribute =
            (PostLogOnAuthenticationSoapExtensionAttribute)initializer;
    }

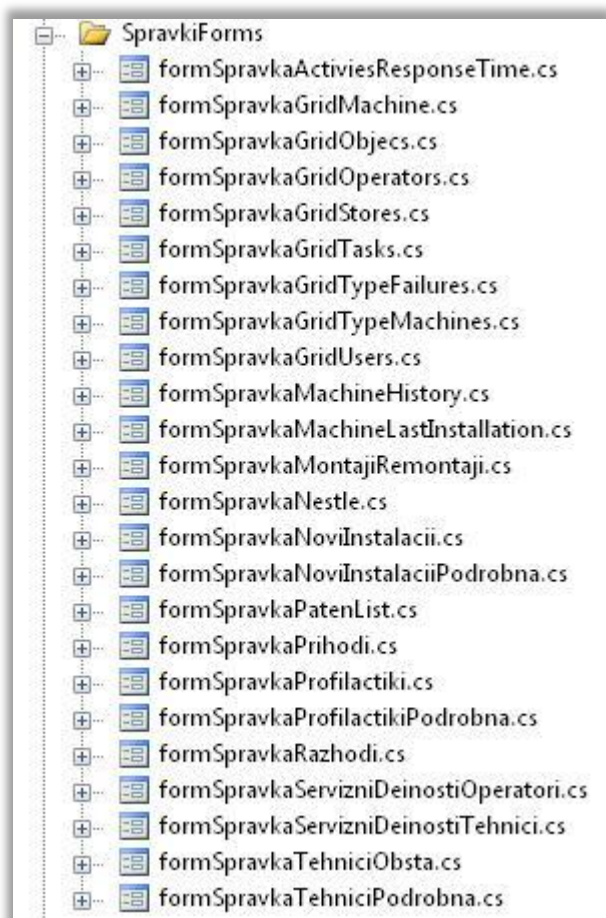
    public override void ProcessMessage(SoapMessage message)
    {
        switch (message.Stage)
        {
            case SoapMessageStage.BeforeSerialize:
            {
                TPostLogOnAuthenticationSoapHeader header =
                    new TPostLogOnAuthenticationSoapHeader();
                header.GUIDint = Program.AdminGuidint;
                message.Headers.Add(header);
            }
            break;
        }
    }
}

```

Основния метод, *ProcessMessage*, се извиква на всяка фаза от обработката на SOAP съобщението. Тук имат значение само *BeforeSerialize* фазата за да се създаден инстанция на *TpostLogOnAutenticationSoapHeader*, и да се присвои стойността за ID-то на потребителската сесия и да се добави в заглавната част на съобщението преди да е сериализирано. По този начин всеки Web метод може да се добавя SOAP header атрибут със съдържат ID-то на потребителската сесия.

3.3. Структура на приложението





фиг. 27 Структура на приложението

V Реализация на базата данни

1. Генериране на кода на базата данни

За създаването и управлението на базата данни се използват СУБД (Системи за управление на бази данни). Всяка СУБД е програмна система, чието предназначение е да създава и управлява данните, организирани в базата от данни. Пълноценното функциониране на една СУБД налага тя да осигурява изискванията към БД:

- разделяне на описанието на данните от тяхната обработка;
- логическа и физическа независимост;
- минимално излишество на данни в БД;
- удобен потребителски интерфейс;
- осигуряване цялостност на данните, т.е. логическа непротиворечивост в БД.

В настоящата дипломна работа за физическата реализация на базата данни е използван SQL Server 2000 на Microsoft. Сървърът поддържа всички важни характеристики на съвременните RDBMS системи (съхранени процедури, транзакции и т.н.). Характерно за него е лесната администрация, подобно на останалите сървъри на Microsoft, добра производителност, висока скалируемост и висока надеждност.

2. Логическа структура на базата данни

База данни е структурирано множество от постоянна информация, съхранявана от компютърна програма. Терминът постоянен означава, че данните продължават да съществуват и след прекратяването на програмата или на потребителската сесия, която ги е създавала. Релационна база данни е съвкупност от такава информация, съхранена в двумерни таблици.

Базата данни на приложението съдържа общо 20 таблици. При проектирането на базата данни са взети под внимание правилата за нормализация. Въпреки това някои от таблиците са денормализирани с цел по-бързото генериране на нужния набор от данни. За да се изпълни изискването за минимален трафик между базата данни и клиентското PocketPC приложение, при обновяване на данните му трябва да се изпращат само новите данни, които са от значение за потребителите на приложението: за операторите – името им, за складовете – град и адрес, за типовете аварии – име, за типовете машини – име и за задачите – тип на дейност, адрес, град и коментар към задачата. За тази цел в таблиците, в които ще се съхраняват данни за оператори, складове, типове аварии и типове машини има колона InsertUpdateIndex.

2.1. Таблици

Използвани съкращения :

- РК – първичен ключ
- FK – външен ключ

Таблица Operators - Съдържа информация за операторите.

Operators			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	Name	nvarchar(100)	No
	MOL	nvarchar(50)	No
	Bulstat	nvarchar(50)	No
	InsertUpdateIndex	bigint	No
	City	nvarchar(50)	No
	Address	nvarchar(255)	No
	Active	bit	No
	Stamp	timestamp	No

фиг. 29 Таблица „Operators”

Полетата в таблицата са следните: **ID (PK)** – идентификатор на оператора; **Name** - име на оператор; **MOL** –имената на материално отговорно лице; **Bulstat** – булстат номер; **InsertUpdateIndex** – индикатор за следене на направените промени върху данните за Name и Active; **City** – град; **Address** – адрес на оператора; **Active** – флаг за активност на оператора; **Stamp** – timestamp за отбелязване на версиите на реда.

Constraints:

- DF_Operators_Active – DEFAULT CONSTRAINT за колоната Active = 1
- IX_Operators_Unique_Name – UNIQUE CONSTRAINT за колоната Name

Indexes:

- IX_Operators_InsertUpdateIndex – индекс за колоната *InsertUpdateIndex*

Triggers:

- INSTEAD_OF_DELETE_OPERATORS – тригера деактивира оператора, т.е *Active* = 0, вместо изтриване на реда.
- INSTEAD_OF_UPDATE_OPERATORS – в тригера се проверява ако се деактивира оператор, се деактивират и всички негови складове и се обновява колоната му *InsertUpdateIndex*, с увеличената отрицателна стойност от колоната *MINOperators* на таблицата *Indexes*. Ако се активира или се променя името на оператора се обновява колоната *InsertUpdateIndex* с увеличената положителна стойност от колоната *MAXOperators* на таблицата *Indexes*. Ако промяната не е нито върху името на оператора, нито върху колоната *Active* се прави обновяване на данните, с тези от таблицата INSERTED, като *InsertUpdateIndex* остава непроменен.
- INSTEAD_OF_INSERT_OPERATORS – при добавяне на нов оператор се взема следващата стойност за *MAXOperators* от таблицат *Indexes*, за да се присвои на колоната *InsertUpdateIndex*. Новия запис се добавя със тази стойност и конверирани към главни букви данни за име на оператор, адрес и град.

Таблица StoreAddress - Съдържа информация за складовете на операторите.

StoreAddress			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	StoreCity	nvarchar(50)	No
	StoreAddress	nvarchar(255)	No
	OperatorID	bigint	No
	InsertUpdateIndex	bigint	No
	Active	bit	No
	Stamp	timestamp	No

фиг. 30 Таблица „StoreAddress”

Има следните полета: **ID (PK)** – идентификатор на склад; **StoreCity** – града, където се намира склада; **StoreAddress** – адреса, на който се намира склада; **OperatorID(FK)** – външен ключ към оператора, на който принадлежи склада; **InsertUpdateIndex** – индикатор за следене на направените промени върху данните за *StoreCity*, *StoreAddress* и *Active*; **Active** – флаг за активност на склада; **Stamp** – timestamp за отбелязване на версиите на реда.

Constraints:

- IX_StoreAddress_Unique_City_Address_OperatorID – UNIQUE CONSTRAINT за колоните *City*, *Address* и *OperatorID*
- DF_StoreAddress_Active – DEFAULT CONSTRAINT за колоната *Active* = 1


Индекси:

- IX_StoreAddress_InsertUpdateIndex – индекс за колоната *InsertUpdateIndex*

Triggers:

- INSTEAD_OF_DELETE_STOREADDRESS - в тригера се деактивира склада, т.е *Active* = 0, вмесо изтриване на реда.
- AFTER_UPDATE_STOREADDRESS - в тригера се проверява ако при обновяването е деактивиран склада, се обновява колоната му *InsertUpdateIndex*, с увеличената отрицателна стойност от колоната *MINStoreAddress* на таблицата *Indexes*. Ако е бил активиран или се е променил града или адреса му се обновява колоната *InsertUpdateIndex* с увеличената положителна стойност от колоната *MAXStoreAddress* на таблицата *Indexes*.
- INSTEAD_OF_INSERT_StoreAddress - при добавяне на нов склад се взема следващата стойност за *MAXStoreAddress* от таблицат *Indexes*, за да се присвои като стойност за *InsertUpdateIndex*. Новия запис се добавя със тази стойност и конверирани към главни букви данни за адрес и град на склад.

Таблица TypeActivity – Съдържа типовете дейности и техните кодове. Те се добавят в базата при създаването ѝ. Не могат да се променят или изтриват.

TypeActivity			
	Column Name	Condensed Type	Nullable
	 ID	bigint	No
	Name	nvarchar(50)	No
	Type	bigint	No

фиг. 31 Таблица „TypeActivity”

Полетата ѝ са: **ID (PK)** – идентификатор на типа дейност; **Name** – името на типа дейност; **Type** – кода на типа дейност.

Constraints:

- IX_TypeActivity_Unique_Name – UNIQUE CONSTRAINT за колоната *Name*.


Indexes:

- IX_TypeActivity_Type – индекс за колоната *Type*.

Triggers:

- INSTEAD_OF_INSERT_UPDATE_DELETE_TypeActivity – върху данните в таблицата не могат да се правят никакви промени, т.е. при добавяне промяна или изтриване на запис тригера не прави нищо.

Таблица TypeMachine - Съдържа информация за типовете машини.

TypeMachine			
	Column Name	Condensed Type	Nullable
	 ID	bigint	No
	Name	nvarchar(50)	No
	InsertUpdateIndex	bigint	No
	Active	bit	No
	Stamp	timestamp	No

фиг. 32 Таблица „TypeMachine”

Тя има следните полета: **ID (PK)** – идентификатор на тип машина; **Name** – име на типа машина; **InsertUpdateIndex** – индикатор за отбелязване на направените промени върху данните за Name и Active; **Active** – флаг за активност на типа машина; **Stamp** – timestamp за отбелязване на версиите на реда.

Constraints:

- IX_TypeMachine_Unique_Name - UNIQUE CONSTRAINT за колоната *Name*
- DF_TypeMachine_Unusable – DEFAULT CONSTRAINT за колоната *Active* = 1

Indexes:

- IX_TypeMachine_InsertUpdateIndex - индекс за колоната *InsertUpdateIndex*.

Triggers:

- INSTEAD_OF_DELETE_TYPERMACHINE – в тригера се деактивира типа машина, т.е *Active = 0*
- AFTER_UPDATE_TYPERMACHINE - в тригера се проверява ако се деактивира типа машина, се обновява колоната му *InsertUpdateIndex*, с увеличената отрицателна стойност от колоната *MINTypeMachine* на таблицата *Indexes*. Ако се активира или се променя името на типа машина се обновява колоната *InsertUpdateIndex* с увеличената положителна стойност от колоната *MAXTypeMachine* на таблицата *Indexes*.
- INSTEAD_OF_INSERT_TYPERMACHINE - при добавяне на нов тип машина се взема следващата стойност за *MAXTypeMachine* от таблицат *Indexes*, за да се присвои на *InsertUpdateIndex*. Новия запис се добавя със тази стойност и конвертирано към главни букви името на типа машина.

Таблица Machine - Съдържа информация за машините.

Полетата ѝ са: **ID (PK)** – идентификатор на машина; **MachineNumber** –номер на машината; **TypeMachineID(FK)** –външен ключ към таблицата с типове на машини, определя типа на машината; **DateInstalation** – датата, на която е била инсталирана машината; **RecipyNumber** – номера на рецептата на машината; **MachineStatus** – статуса на машината, може да бъде една от следните стойности: 1 – монтирана машина; 2 – демонтирана машина; 3 – нова машина; 4- не активна машина; 5 – оставена в склад; 6 – взета от склад; **Season** – указва се дали е сезонна или целогодишна машината; **Contract** – номер на договор; **ReceiveDate** – дата на получаване; **LastPayment** – датата на последната вноска; **PlanNumber** – номер на план; **DrawOutDate** – дата на изтегляне на машината от склада; **StoreAddressID(FK)** – външен ключ към таблицата със складове, т.е. сочи към склада от където е била взета машината; **Stamp** – timestamp за отбелязване на версиите на реда.

Machine			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	MachineNumber	nvarchar(50)	No
	TypeMachineID	bigint	No
	DateInstalation	smalldatetime	Yes
	RecipyNumber	nvarchar(50)	Yes
	MachineStatus	int	No
	Season	bit	No
	Contract	int	Yes
	ReceiveDate	smalldatetime	Yes
	LastPayment	smalldatetime	Yes
	PlanNumber	int	Yes
	DrawOutDate	smalldatetime	Yes
	StoreAddressID	bigint	Yes
	Stamp	timestamp	No

фиг. 33 Таблица „Machine”

Constraints:

- DF_Machine_MachineStatus - DEFAULT CONSTRAINT за колоната *MachineStatus = 1*
- DF_Machine_Season - DEFAULT CONSTRAINT за колоната *MachineStatus = 0*

Indexes: IX_Machine_MachineStatus - индекс за колоната MachineStatus.

Таблица Personnel - Съдържа данни за потребителите в системата.

Полетата са: **ID (PK)** – идентификатор на потребител; **Password** – хеш стойността на паролата на потребителя за достъп до системата; **PocketPCID** – уникален идентификатор на PocketPC апарата; **Name** – имената на потребителя; **PersonnelNumber** – личния номер на потребителя; **TelephoneNumber** – телефон на потребителя; **Active** – флаг за активност на потребителя; **GUIDint** – идентификатор за потребителска сесия; **Address** – адрес на потребителя; **City** – град на потребителя; **Stamp** – timestamp за отбелязване на версиите на реда.

Personnel			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	Password	nvarchar(50)	No
	PocketPCID	nvarchar(255)	Yes
	Name	nvarchar(255)	No
	PersonnelNumber	varchar(10)	No
	TelephoneNum...	nvarchar(50)	Yes
	Active	bit	No
	GUIDint	int	Yes
	Address	nvarchar(255)	Yes
	City	nvarchar(50)	Yes
	Stamp	timestamp	No

фиг. 34 Таблица „Personnel”

Constraints:

- IX_Personnel_UNIQUE_Password_PocketPCID – UNIQUE CONSTRAINT за колоните *Password* и *PocketPCID*.
- IX_UNIQUE_GUIDint – UNIQUE CONSTRAINT за колоната *GUIDint*.

Индекси:

- IX_Personnel - индекс за колоната *PersonnelNumber*.

Triggers:

- INSTEAD_OF_DELETE_PERSONNEL - в тригера се деактивира потребителя, т.е *Active* = 0
- INSTEAD_OF_UPDATE_Personnel - в тригера се проверява ако се обновяват данни за потребител, който е администратор, стойността за *PocketPCID* винаги се взема от функцията *AdminGUID*, за да се осигури че не е променена. Всички останали данни се обновяват.
- INSTEAD_OF_INSERT_Personnel – в тригера се прави запис с новите данни като данните за *Name*, *Address*, *City* се конвертира към главни букви.

Таблица TypeFailure - Тази таблица съдържа информация за типовете аварии.

TypeFailure			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	Name	nvarchar(255)	No
	InsertUpdateIndex	bigint	No
	Active	bit	No
	Stamp	timestamp	No

фиг. 35 Таблица „TypeFailure”

Тя съдържа полетата **ID (PK)**– идентификатор на типа авария; **Name** – име на тип авария; **InsertUpdateIndex** – индикатор за отбелязване на направените промени върху името на авария; **Active** – флаг за активност на типа авария; **Stamp** – timestamp за отбелязване на версиите на реда.

Constraints:

- IX_TypeFailure_Unique_Name - UNIQUE CONSTRAINT за колоната *Name*.
- DF_TypeFailure_Active – DEFAULT CONSTRAINT за колоната *Active* = 1

Индекси:

- IX_TypeFailure_InsertUpdateIndex – индекс за колоната *InsertUpdateIndex*.

Triggers:

- INSTEAD_OF_DELETE_TypeFailure – в тригера се деактивира типа авария, т.е *Active* = 0.
- AFTER_UPDATE_TypeFailure - в тригера се проверява ако се деактивира типа авария, се обновява колоната *InsertUpdateIndex* с увеличената отрицателна стойност от колоната *MINTypeFailure* на таблицата *Indexes*. Ако се активира или се промена името на типа авария се обновява колоната *InsertUpdateIndex* с увеличената положителна стойност от колоната *MAXTypeFailure* на таблицата *Indexes*.
- INSTEAD_OF_INSERT_TypeFailure - при добавяне на нов тип авария се взема следващата стойност за *MAXTypeFailure* от таблицата *Indexes*, за да се сложи като стойност за *InsertUpdateIndex*. Новия запис се добавя със тази стойност и конвертирано към главни букви името на типа авария.

Таблица FailureVisit - Таблицата съхранява допълнителна информация, към информацията в таблица *Activities*, за извършените дейности от тип Авария.

FailureVisit			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	TypeFailureID	bigint	Yes
	Comment	nvarchar(255)	Yes

фиг. 36 Таблица „FailureVisit”

Полетата ѝ са: **ID (PK)** – идентификатор на дейността; **TypeFailureID(FK)** – външен ключ към таблицата с типове аварии.; **Comment** – коментар към направената авария.

Таблица OtherVisit – Таблицата съхранява допълнителна информация, към информацията в таблица *Activities*, за извършените дейности от тип Служебно Друго.

OtherVisit			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	Description	nvarchar(4000)	No
	Price	float	Yes
	Type	int	No
	FactureNumber	nvarchar(50)	Yes

фиг. 37 Таблица „OtherVisit”

Полетата са: **ID (PK)** – идентификатор на дейността; **Description** – описание на извършената дейност; **Price** – стойност на направени разходи за дейността; **Type** – тип на направената дейност, може да бъде: 1 – служебно друго или 2 – служебно друго с направен разход; **FactureNumber** – номер на фактура;

Таблица InstallVisit - Таблицата съхранява допълнителна информация, към информацията в таблица *Activities*, за извършените дейности от тип Монтаж.

InstallVisit			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	TypeInstalation	int	Yes
	Price	float	Yes
	OperatorID	bigint	Yes
	KreditKontrol	bit	Yes
	MonetenMehanizam	bit	Yes

фиг. 38 Таблица „InstallVisit”

Таблицата има полета: **ID (PK)** – уникален идентификатор на дейността; **TypeInstallation** – код на типа инсталация; **Price** – цена на инсталацията; **OperatorID (FK)** – външен ключ към таблицата с оператори, т.е. за кой оператор се отнася направената инсталация; **KreditKontrol** – отбелязва се дали машината е на кредитен контрол; **MonetenMehanizam** – флаг за отбелязване дали машината е монтирана с монетен механизъм.

Таблица NotServiceVisit – Таблицата съхранява допълнителна информация, към информацията в таблица *Activities*, за извършените дейности от тип Несервизно посещение.

NotServiceVisit			
	Column Name	Condensed Type	Nullable
	ID	bigint	No
	Description	nvarchar(4000)	No
	Price	float	No

фиг. 39 Таблица „NotServiceVisit”

Полетата й са следните: **ID (PK)** – уникален идентификатор на дейността; **Description** – описание за несервизното посещение; **Price**- стойност на направения разход при посещението.

Таблица FuelVisit – Таблицата съхранява допълнителна информация, към информацията в таблица *Activities*, за извършените дейности от тип Посещение гориво.

FuelVisit			
	Column Name	Condensed Type	Nullable
	ID	bigint	No
	Type	int	No
	Kg	float	No
	Price	float	No

фиг. 40 Таблица „FuelVisit”

Полетата й са следните: **ID (PK)** – уникален идентификатор на дейността; **Type** – код на типа гориво; **Kg** – стойността на заредените килограми; **Price**- стойност на зареденото гориво.

Таблица ProfilacticVisit – Таблицата съхранява допълнителна информация, към информацията в таблица *Activities*, за извършените дейности от тип Профилактика.

ProfilacticVisit			
	Column Name	Condensed Type	Nullable
	ID	bigint	No
	Type	int	No
	Price	float	Yes

фиг. 41 Таблица „ProfilacticVisit”

Таблицата съдържа полетата: **ID (PK)** – уникален идентификатор на дейността; **Type** – код на типа профилактика, може да бъде: 1 – механична или 2 - химична; **Price**- стойност на направения разход при посещението.

Таблица StoreVisit – В нея се съхранява допълнителна информация, към информацията в таблица *Activities*, за извършените дейности от тип Посещение склад.

StoreVisit			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	StoreAddressID	bigint	Yes
	Description	nvarchar(255)	Yes

фиг. 42 Таблица „StoreVisit”

Полетата ѝ са: **ID (PK)** – уникален идентификатор на дейността; **StoreAddressID(FK)** – външен ключ към таблицата със складове, т.е сочи към склада където е извършена дейността; **Description** – описание на извършената дейност.

Таблица MachineAddresses – Служи за съхранение на адресите, където са се намирали машините.

MachineAddress			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	MachineID	bigint	No
	ObjectName	nvarchar(100)	No
	City	nvarchar(50)	Yes
	Address	nvarchar(255)	Yes
	ObektID	bigint	Yes

фиг. 43 Таблица „MachineAddress”

Има следните полета : **ID (PK)** – уникален идентификатор на адрес; **MachineID(FK)** външен ключ към таблицата с машини, т.е. указва за коя машина се отнася записа; **ObjectName** – името на обекта, където се намира машината; **City** – града на обекта; **Address** – адреса на обекта; **ObektID(FK)** – външен ключ към таблицата с обекти.

Индекси:

- IX_MachineAddress - индекс за колоната MachineID..

Triggers:

- INSTEAD_OF_UPDATE_MachineAddress,
- INSTEAD_OF_INSERT_MachineAddress и в двата тригера се обновяват, съответно добавят данните от таблицата INSERTED като данните за *ObjectName*, *City* и *Address*, се конвертират към големи букви.

Таблица Tasks – В тази таблица се съхранява информация за задачите, които техниците трябва да извършат.

Tasks			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	PersonnelID	bigint	No
	ObektID	bigint	No
	TypeActivityID	bigint	No
	Active	bit	No
	Finished	bit	No
	StartDate	smalldatetime	No
	EndDate	smalldatetime	Yes
	InsertUpdateIndex	bigint	No
	Comment	nvarchar(100)	Yes
	AssignedFrom	nvarchar(100)	Yes
	Stamp	timestamp	No

фиг. 44 Таблица „Tasks”

Полета са: **ID (PK)** – уникален идентификатор на задача; **PersonnelID(FK)** – външен ключ към таблицата с техници, т.е. указва за кой техник е предназначена задачата; **ObektID(FK)** – външен ключ към таблицата с обект, т.е. показва къде трябва да се извърши задачата; **TypeActivityID(FK)** – външен ключ към таблицата с типове дейност; **Active** – флаг за указване дали е активна задачата; **Finished** – флаг за указване дали е приключена задачата; **StartDate** – начало на задаване на задачата от администратора; **EndDate** – датата на приключване на задачата; **InsertUpdateIndex** - индикатор за отбелязване на направените промени върху данните; **Comment** – коментар към задачата; **AssignedFrom** – информация за заявителя; **Stamp** – timestamp за отбелязване на версиите на реда.

Constraints:

- DF_Tasks_Active - DEFAULT CONSTRAINT за колоната *Active* = 1.
- DF_Tasks_Finished – DEFAULT CONSTRAINT за колоната *Finished* = 0

Trrigers:

- INSTEAD_OF_DELETE_Tasks – в тригера се проверява ако задачата е приключена не се извършва нищо, в противен случай в тригера деактивира задачата, т.е *Active* = 0.
- INSTEAD_OF_UPDATE_Tasks - в тригера се проверява ако задачата е приключена не се извършват промени върху записа. Ако се деактивира или се приключва задача, се обновява колоната за *InsertUpdateIndex*, с увеличената отрицателна стойност от колоната *MINTasks* на таблицата *Indexes*. Ако се активира задачата или просто се променят някои от останалите данни се обновява колоната *InsertUpdateIndex* с увеличената положителна стойност от колоната *MAXTasks* на таблицата *Indexes*, а останалите данни се взимата от таблицата INSERTED .
- INSTEAD_OF_INSERT_Tasks - при добавяне на нова задача се взима следващата стойност за *MAXTasks* от таблицат *Indexes*, за да се присвои

на *InsertUpdateIndex*. Новия запис се добавя със тази стойност, а останалите данни се взимат от таблицата INSERTED.

Таблица Objects – В тази таблица се съхранява информация за обектите, на които се монтират машини.

Objects			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	Name	nvarchar(100)	No
	Address	nvarchar(255)	No
	City	nvarchar(50)	No
	Region	nvarchar(50)	No
	Firm	nvarchar(100)	Yes
	FirmAddress	nvarchar(255)	Yes
	MOL	nvarchar(50)	Yes
	EGN	nchar(10)	Yes
	Bulstat	nchar(20)	Yes
	Obligations	nvarchar(30)	Yes
	Dealer	nvarchar(50)	Yes
	Stamp	timestamp	No
	Active	bit	No

фиг. 45 Таблица „Object”

Поleta са: **ID (PK)** – уникален идентификатор на обект; **Name** – името на обекта; **Address** – адреса на който се намира обекта; **City** – града на обекта; **Region** – района, в който се намира града на обекта; **Firm** – името на фирмата; **FirmAddress** – адрес на фирмата; **MOL** – МОЛ на фирмата; **EGN** – ЕГН на МОЛ-а; **Bulstat** – булстат на фирмата; **Obligations** – информация за договорки; **Dealer** – търговец с който е сключен договора; **Stamp** – timestamp за отбелязване на версиите на реда; **Active** – флаг за указване на активност на обекта.

Constraints:

- DF_Objects_Active - DEFAULT CONSTRAINT за колоната *Active* = 1.
- IX_Objects_Unique_Name – UNIQUE CONSTRAINT за колоната *Name*.

Triggers:

- INSTEAD_OF_DELETE_Objects – в тригера се деактивира обекта, т.е. *Active* = 0.
- INSTEAD_OF_UPDATE_Objects – в тригера се проверява ако се деактивира обект, дали има монтирани машини на него. Ако има такива обекта не може да се деактивира. В противен случай се обновяват данните с тези от таблицата INSERTED.
- INSTEAD_OF_INSERT_Objects – Всички данни, с изключение на тези за *Obligations*, се конвертира към главни букви преди да се добави записа.

Таблица Activities – В таблицата се съхраняват данни за всички дейности извършени от потребители.

Полета са: **ID (PK)** – уникален идентификатор на дейност; **PersonnelID(FK)** – външен ключ към таблицата с потребители; **MachineID(FK)** – външен ключ към таблицата с машини; **TypeActivityID(FK)** – външен ключ към таблицата с типове дейности; **Date** – дата, на която е извършена дейността; **ForeignID(FK)** – външен ключ към таблиците за типове дейности; **Kilometers** – километраж при извършване на дейността; **MachineAddressID(FK)** – външен ключ към таблицата с адреси на машини; **PassKilometers** – изминатите километри от предходната дейност, **TaskID(FK)** – външен ключ към таблицата със задачи, т.е. дали дейността е свързана с поставена задача; **ActivityGUID** – уникален идентификатор на дейността.

Activities			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	PersonnelID	bigint	No
	MachineID	bigint	Yes
	TypeActivityID	bigint	No
	Date	smalldatetime	Yes
	ForeignID	bigint	Yes
	Kilometers	int	Yes
	MachineAddressID	bigint	Yes
	PassKilometers	int	Yes
	TaskID	bigint	Yes
	ActivityGUID	uniqueidentifier	No

фиг. 46 Таблица „Activities”

Indexes:

- IX_Activities_MachineID - индекс за колоната *MachineID*.
- IX_Activities_Date – индекс за колоната *Date*.
- IX_Activities_TypeActivityID – индекс за колоната *TypeActivityID*.

Таблица Indexes – Таблицата съдържа минималните и максималните стойности на *InsertUpdateIndex* колоните съответно за таблиците Operators, StoreAddress, TypeFailure, TypeMachine и Tasks. При създаване на базата данни в таблицата се добавя един ред с стойност 0 във колоните. Този ред не може да се тире и не могат да се добавят нови редове.

Полета са: **ID (PK)** – уникален идентификатор на обект; **BGDate** – часовата разликата между България и сървъра; **MAXOperators** – максималната стойност в колоната *InsertUpdateIndex* за таблицата *Operators*; **MINOperators** – минималната стойност в колоната *InsertUpdateIndex* за таблицата *Operators*; и т.н за всяка от таблиците.

Indexes			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	BGDate	bigint	Yes
	MAXOperators	bigint	No
	MINOperators	bigint	No
	MAXStoreAddress	bigint	No
	MINStoreAddress	bigint	No
	MAXTypeFailure	bigint	No
	MINTypeFailure	bigint	No
	MAXTypeMachine	bigint	No
	MINTypeMachine	bigint	No
	MAXTasks	bigint	No
	MINTasks	bigint	No

фиг. 47 Таблица „Indexes”

Constraints:

- DF_Indexes_MAXOperators – DEFAULT CONSTRAINT за колоната *MAXOperators* = 0.

- DF_Indexes_MINOperators – DEFAULT CONSTRAINT за колоната *MINOperators* = 0 и т.н. за всяка от колоните.

Triggers:

- INSTEAD_OF_DELETE_INDEXES – тригера не допуска изтриване на запис.
- INSTEAD_OF_UPDATE_INDEXES – в зависимост от колоната, чиято стойност се променя, се увеличава с 1 или се намаля с 1 старата стойност.
- INSTEAD_OF_INSERT_INDEXES – тригера не допуска добавяне на запис, тъй като трябва да се поддържа само един запис в таблицата.

Таблица RouteByDay – В таблицата се съхраняват данни за маршрута на всеки потребител за всеки отделен ден и личните му километри за този ден.

RouteByDay			
	Column Name	Condensed Type	Nullable
🔑	ID	bigint	No
	PersonnelID	bigint	No
	Route	nvarchar(4000)	Yes
	Date	smalldatetime	No
	PersonnelKilometers	int	No
	LastCity	nvarchar(50)	Yes

фиг. 48 Таблица „RouteByDay”

Полета са: **ID (PK)** – уникален идентификатор; **PersonnelID(FK)** – външен ключ към таблицата с потребители; **Route** – маршрута, на потребителя за деня; **Date** – дата за която се отнася запис; **PersonnelKilometers** – личните километри за този ден на потребителя; **LastCity** – последния град, през който е минал потребителя.

Constraints:

- DF_RouteByDay_PersonnelKilometres - DEFAULT CONSTRAINT за колоната *PersonnelKilometres* = 0

Indexes:

- IX_RouteByDay – UNIQUE CONSTRAINT за колоната *PersonnelID*.
- IX_RouteByDay_1 - UNIQUE CONSTRAINT за колоната *Date*.

VI Внедряване

1. Fonte Fresco Mobile Cab Package

Приложението включва две стъпки за да се инсталира правилно на мобилно PDA устройство. Копира се cab файл на устройството и се стартира ръчно от администратора.

2. Fonte Fresco Service MSI

Администраторското приложение се инсталира на компютър с операционна система Windows XP Service Pack 2 като единствената предпоставка е инсталиран .NET Framework v.2.0. Инсталацията е автоматизирана и не изисква намесата на администратор.

3. Fonte Fresco Web Service MSI

Fonte Fresco Web Service включва проверка на следните стартови условия (*launch conditions*) за да се инсталира успешно:

- Инсталиран .Net Framework v.1.1.4322:
.NET Framework ver.1.1.4322
- Инсталиран Internet Information Services 5.0 или по-висока:
IISVersion >= "#5"
- Инсталацията е стартирана от потребител от групата на администраторите
Priviledged = "1"
- Текущата версията на Windows е Windows 2003 Service Pack 1 или по-висока:
VersionNT = 502 and ServicePackLevel >= 1

Преди да се изчислят стартовите условия MSI engine изчислява дали са достъпни стартовите предпоставки (*prerequisites*). Инсталиран .NET v2.0, Window Installer 3.1, SQL Server 2005 Express Edition Service pack 1. Ако някои предпоставка не е налице тя се сваля от официалната страница на Microsoft и се инсталира автоматично без да е необходима намесата на администратор.

След инсталация на всички файлове се преминава финална настройка на правата на някои директории от инсталацията.

- Настройват се правата на избраната от потребителя директория със системния акаунт NETWORKSERVICE за да може SQL Server Express да има достъп до базата данни на Web услугата. Стартира се:
cacls.exe [TARGETDIR]\APP_DATA /t/e /p NETWORKSERVICE:F
- Настройват се правата на Log директорията със системния акаунт NETWORKSERVICE за да може Internet Information Services да има достъп до нея. Стартира се:
cacls.exe [TARGETDIR]\Log /t/e /p NETWORKSERVICE:F
- Настройват се атрибутите на инсталираните файлове в избраната от потребителя директория. Стартира се:
attrib.exe -R -S -A -H [TARGETDIR].*/S/D*

- Настройва се web.config файла на Web услугата. Стартира се специално създадено за целта приложение:

XmlEdit.exe [TARGETDIR]Web.config "Data Source=.\SQLEXPRESS;integrated security=true;attachdbfilename=[TARGETDIR]App_Data\FonteFrescoServices.mdf;"

VII Процес на разработка

При разработката на информационната система не е приложен традиционния водопаден модел за разработка на софтуер. Използван е итерационен модел на софтуерния процес, най – вече поради непълната яснота по отношение на функционалността, която трябва да предостави системата. Процеса на разработка трябваше да бъде достатъчно гъвкав за да може да се удовлетворят изисквания на клиента, поставени в късен период от разработката на системата. Основните принципи които са следвани в процеса на разработка на системата са:

- удовлетворяване на клиентските изисквания
- въвличане на представител от страна на клиента в разработката. По този начин клиента по-лесно си изяснява какви изисквания има към системата, разработчиците не правят предположения за това какво би искал клиента(често са погрешни), бързият callback от клиента предотвратява риска от разработка на софтуер, който не отговаря на потребителските изисквания.
- проста архитектура, за да може да се имплементират лесно промени в изискванията или нови такива, колкото и късно да са поставени в процеса на разработка.
- да има интуитивен потребителски интерфейс

Ръководени от тези принципи избора ни за модел на процес на разработка бе насочен към Гъвкавите методологии, в частност към *XP(Extreme Programming)*. В много от случаите на прилагане на *XP*, някои от практиките, заложи в методологията, се модифицират за да се постигната максимално добри резултати за конкретната разработка на софтуерен продукт. В процеса на разработка на информационната система, разглеждана в дипломната работа, са приложени следните практики:

- *On-site customer* – Включване на клиента в екипа на живо и непрекъснато, за да отваря на въпроси.
Приложение: В екипа участваше представител от страна на клиента, който беше непрекъснато на разположение за отговори на въпроси, но не на живо. Срещи на живо се правеха веднъж на 1-2 седмици, за да се дефинират *stories* за следващата итерация.
- *Planning Game* – Бързо определяне на обхвата на следващата итерация, комбинирайки бизнес приоритетите и техническите оценки.
Приложение: Клиента разказва *stories*(потребителски случаи на употреба), които приложението трябва да изпълнява. Тези *stories* са оценени според трудността.
- *Small releases* – Поставяне на система в продукцията бързо, след това пускане нова версия за много кратък цикъл.
Приложение: Предаване на release на 3-4 месец; кратки итерации 1-3 седмици.
- *Coding standards* – Разработчиците пишат целия код в съответствие с правила, значими за комуникация чрез кода.

Приложение: Използване на региони (*#region*) за отделяне логически частите на класовете; конвенция за именуване: за константи(пр. `ERROR_INTERNET_CONNECTION_TEXT`), за локални променливи(`_username`) и т.н.

- *Pair programming* – Целият код се пише от двама разработчици на една машина.

Приложение: Не целият код, но част от него е разработена от двама разработчици едновременно.

- *Simple design* – Дизайна на системата трябва да бъде колкото се може по опростен във всеки един момент.

Приложение: Системата се имплементира с не усложнен код, който да изпълнява изискваните функционалности.

- *Collective ownership* – Всеки може да променя който и да е код от системата по всяко време.

Приложение: Всеки от разработчиците е отговорен за цялата система, познава системата, и знае какъв ще е ефекта върху системата, когато прави промяна.

- *Refactoring* – Разработчиците реструктурират системата, без да променят логиката, за да премахнат повторения, подобрят производителността или за да добавят гъвкавост.

Приложение: Използва се през целия процес на разработка за подобряване на качеството на кода.

- *Continuous integration* – Интегриране и `build` на системата няколко пъти на ден, всеки път след приключване на задача.

Приложение: След приключване на задача или направена промяна, се *build*-ва приложението и се прави тест дали задачата или промяната предоставя изискваната функционалност.

При разработката е използвана системата за управление и контрол на софтуера при разработка *Borland StartTeam*.

Заклучение

В дипломната работа са разгледани технологиите .NET и .NET CompactFramework, възможностите и за работа с бази данни и предимствата на трислойния модел на комуникация. Представена е трислойната архитектура на създаденото приложение, като клиентския слой и слой за база данни са разгледани детайлно. Описан е процеса на разработка на информационната система.

На базата на потребителските изисквания са изведени случаи на употреба по методологията на UML. Така дефинираният модел е имплементиран, за да се получи информационна система, която напълно отговаря на изискванията за контролиране, регистриране и отчитане на дейността на техниците във фирмата. Информационната система е проектирана, така че нейната интеграция да укаже максимална съвместимост с изградения процес на работа. Чрез разработената система ръководството получава ежеминутна информация за всяка една дейност извършена от техниците. Чрез интуитивния си интерфейс клиентските приложения позволяват на всеки служител, изключително бързо да се запознае с възможностите и начина на работа.

Благодарение на използвания итеративен подход за разработка и някои от практиките на Extreme Programming модела, някои рискове като изграждане на приложения, които не отговарят на потребителските изисквания, или невъзможност за реализиране на допълнителни изисквания, освен първоначалните, или сложна софтуерна архитектура, която е труден за поддръжка бяха избегнати.

Възможностите за бъдещо развитие на системата са изцяло свързани и зависят от бизнес нуждите на фирмата. Една такава възможност е добавянето на графики към справките в администраторското приложение, което би дало по-ясно и по-цялостно представяне на извлечената информацията.

Използвана литература

1. <http://msdn2.microsoft.com>
2. <http://msdn.microsoft.com/practices>
3. „Програмиране за .NET Framework” – Светлин Наков и колектив
4. „Design Patterns Елемнти на обектно-ориентирания софтуер за многократно използване” – Ерика Гама, Ричард Хелм, Ралф Джонсън, Джон Влисидец
5. “The Gurus Guide To Transact - SQL” – Ken Henderson, 2000
6. “Pro C# 2005 and the .NET 2.0 Platform” – Andrew Troelsen
7. Курс по „Гъвкави методологии” – Силвия Илиева
8. „The Unified Software Development Process. Addison-Wesley” - Ivar Jacobson, Grady Booch, James Rumbaugh.
9. „Microsoft® .NET Compact Framework Kick Start” – Erik Rubin, Ronnie Yates

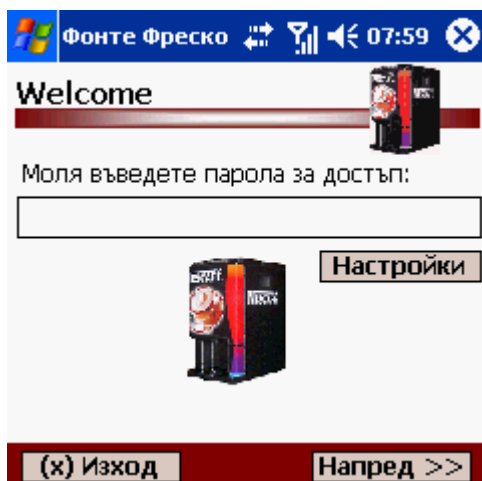
Приложение А Интерфейс на FonteFrescoMobile

В приложението се описва последователността от форми през която минава потребителя при потребителския случай на *online* въвеждане на данни за дейност от тип Монтаж.



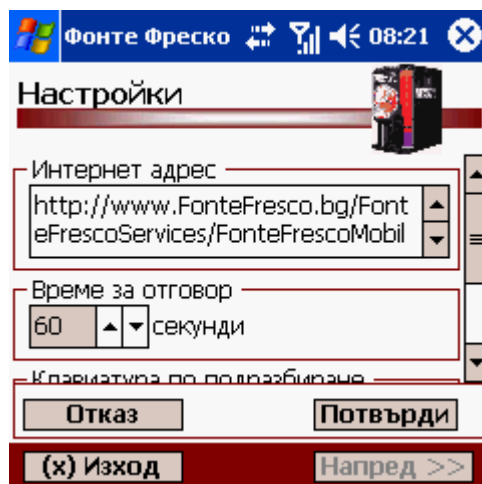
фиг. 49 „Splash” форма

При стартиране на приложението се показва „Splash” форма докато се инициализира формата „Вход в системата” и се заредят настройките на приложението и данните от XML файла.



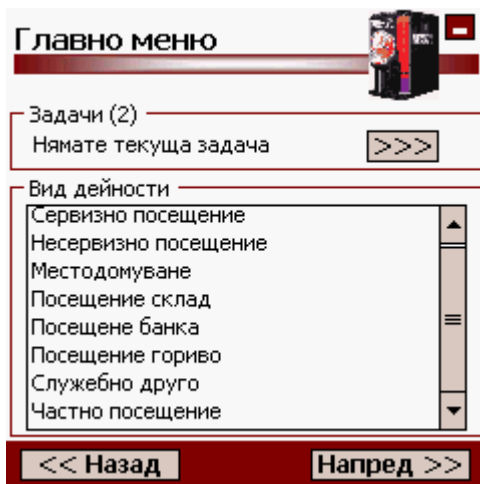
фиг. 50 Форма „Вход в системата”

Ако приложението няма въведени настройки техника ги въвежда във следващата форма, която се показва при натискане на бутона „Настройки”



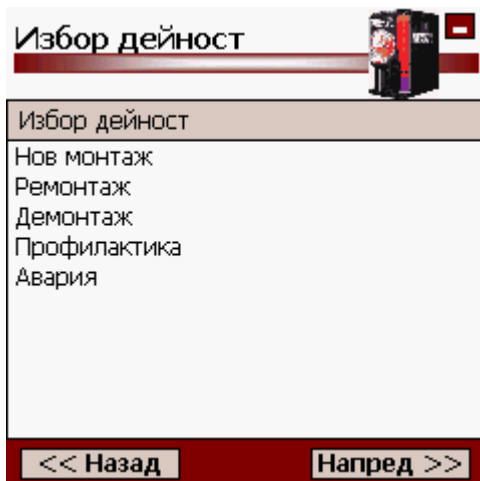
фиг. 51 Форма „Настройки”

След натискане на бутона „Потвърди”, настройките се съхраняват в *FonteFrescoMobile.settings* файла и на потребителя отново се показва форма „Вход в системата”



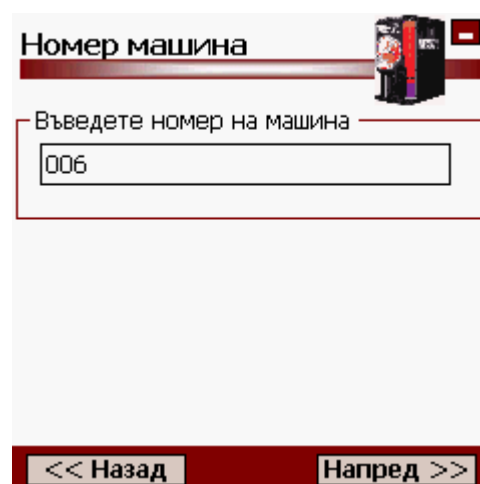
фиг. 52 Форма „Главно меню”

Формата предоставя възможност на техника да избере задача за дейността. Контрола за прегледани и избор на задачи е показан на фиг. 17. При избиране на *Сервизно посещение* и натискане на бутона „*Напред*” на потребителя се показва следващата форма.



фиг. 53 Форма „Сервизна дейност”

При online режима на работа на приложението избраната дейност от списъка трябва да отговаря на дейността указана в асоциираната с нея задача, в противен случай потребителя не може да премине напред. При избиране на *Нов монтаж* на потребителя се показва следната форма.



фиг. 54 Форма „Номер на машина”

За да се валидира въведения номер се прави обръщение към Web услугата и ако е валиден се получават данни необходими за въвеждания тип дейност за машината.

Данни за машина

Обект: КАФЕ ДЖУРАСИК

Град/село: ВЕЛИКО ТЪРНОВО

Адрес: УЛ. НИКОЛА ГАБРОВСКИ 43

Рецепта №:

Оператор: МАКСИМ

<< Назад Напред >>

Данни за машина

Рецепта №:

Оператор: МАКСИМ

Модел: CAFFE 05

Кредит контрол

Монетен механизъм

<< Назад Напред >>

фиг. 55 Форма „Данни за машина”

След успешна валидация на номера на машината на потребителя се изпращат данните за *Обект*, *Град/село*, *Адрес*, *Оператор* и *Модел* на машина. Останалите данни се въвеждат от техника.

Вид монтаж

Вид: комбиниран

сума: 34 лв.

водопровод и помпа

водопровод и доливане

<< Назад Напред >>

фиг. 56 Форма „Вид монтаж”

След като е въвел данните за машината при сервизна дейност от тип монтаж или ремонт, потребителят трябва да укаже и вида монтаж във тази форма. В зависимост от избрания вид монтаж, потребителят трябва да въведе допълнителна информация в появилите се полета под падащия списък.

Километраж

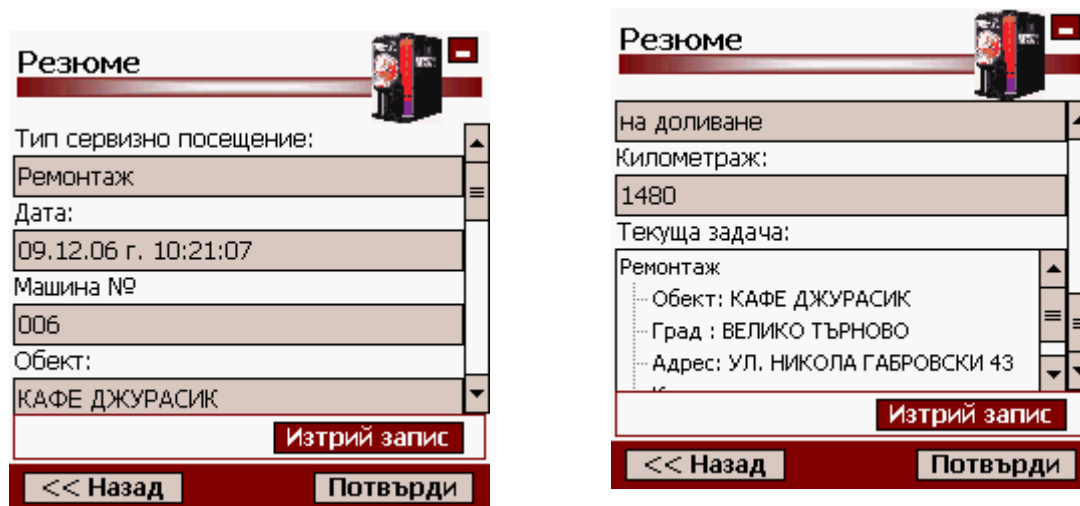
Въведете километраж: 56888

<< Назад Напред >>

фиг. 57 Форма „Километраж”

След като е въвел необходимите данни, според типа на избраната дейност, потребителят трябва да въведе и километража на автомобила. Въведения километраж трябва да бъде между 3 и 10 цифри

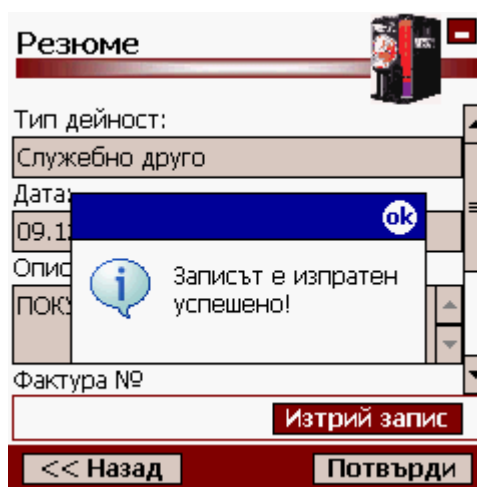
Когато потребителят преглежда запис, който е бил съхранен в апарата, той не може да редактира въведеният преди това километраж.



фиг. 58 Форма „Резюме”

Формата представя цялата информация, въведената от потребителя, за конкретната дейност. Тук потребителят има възможност да изтрие въведените данни за избраната дейност.

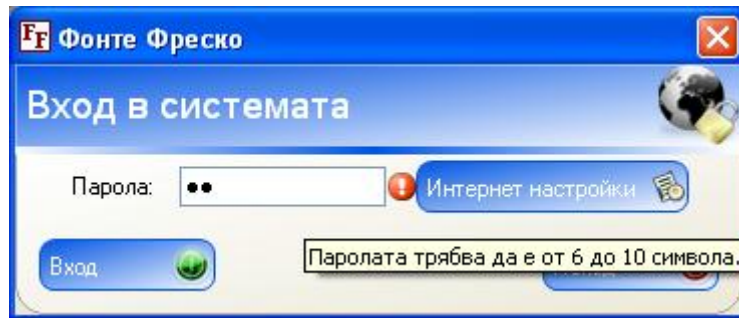
За да съхрани записът на сървъра, потребителят трябва да натисне бутона „Потвърди”. При успех той ще види следното съобщение:



фиг. 59 Форма „Резюме”

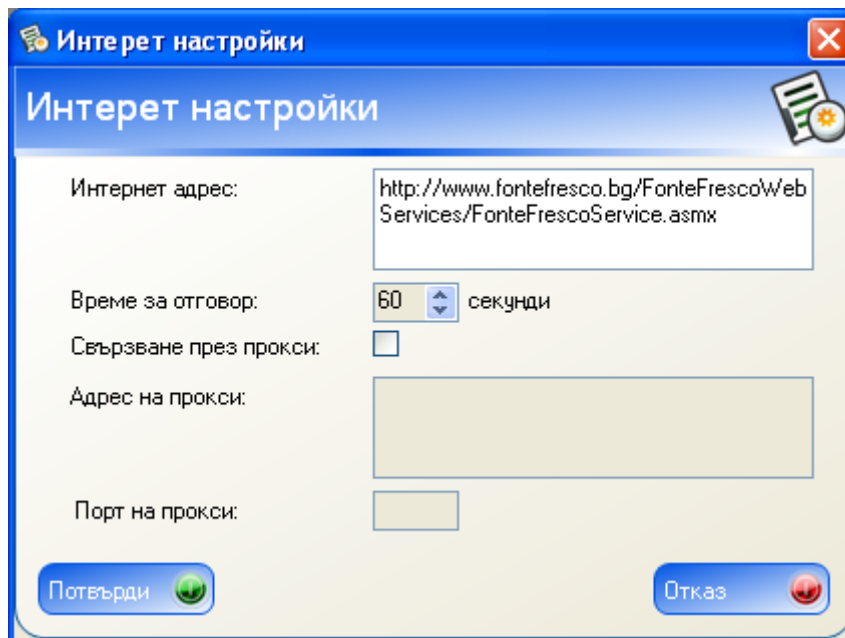
Приложение В Интерфейс на FonteFrescoService

В приложението са показани някои от формите на FonteFrescoService.



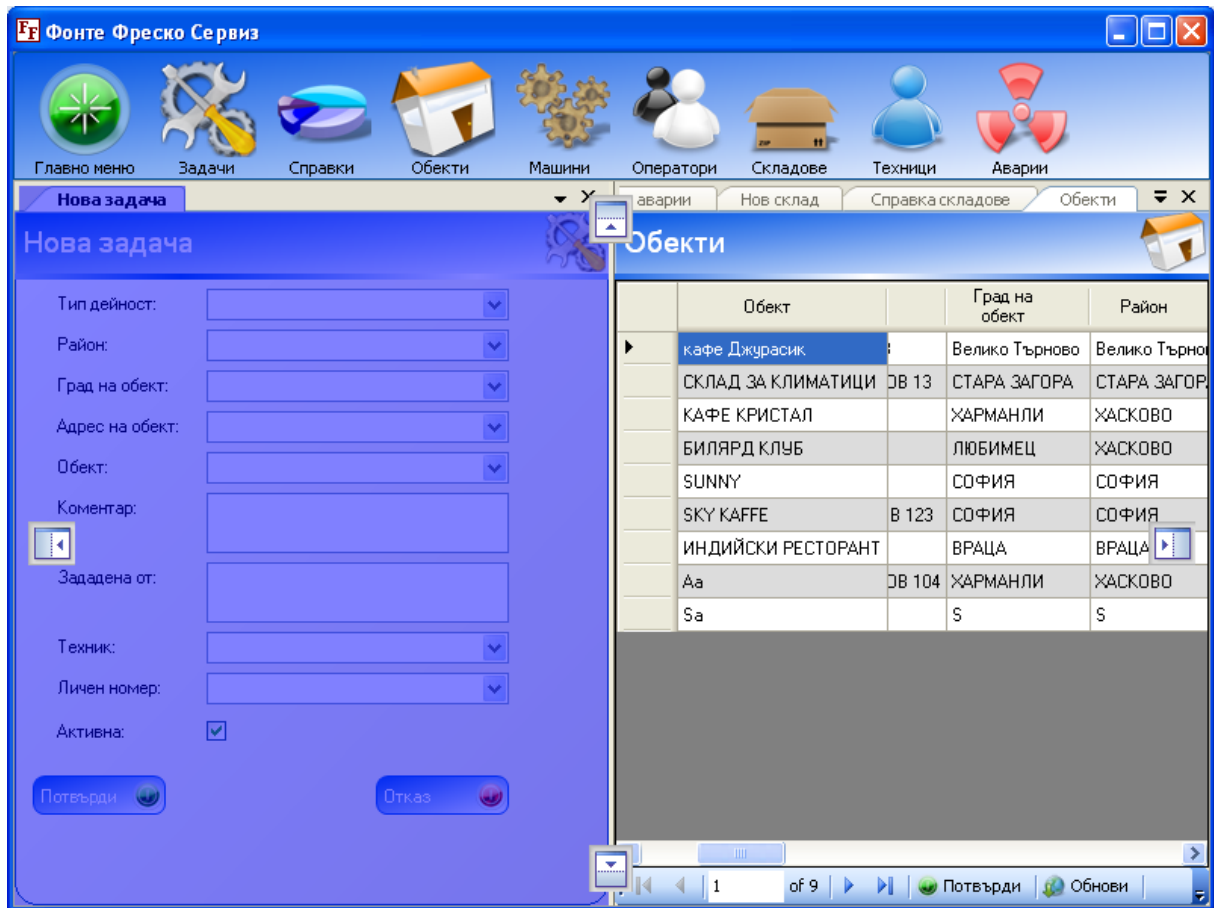
The screenshot shows a window titled "Фонте Фреско" (Fonte Fresco) with a close button in the top right corner. The main title is "Вход в системата" (Login to the system). Below the title is a password input field labeled "Парола:" with two black dots representing the password. To the right of the password field is a button labeled "Интернет настройки" (Internet settings) with a gear icon. Below the password field is a "Вход" (Login) button with a green arrow icon. A tooltip message is displayed over the bottom right of the form, stating "Паролата трябва да е от 6 до 10 символа." (The password must be 6 to 10 characters long).

фиг. 60 Форма „Вход в системата”

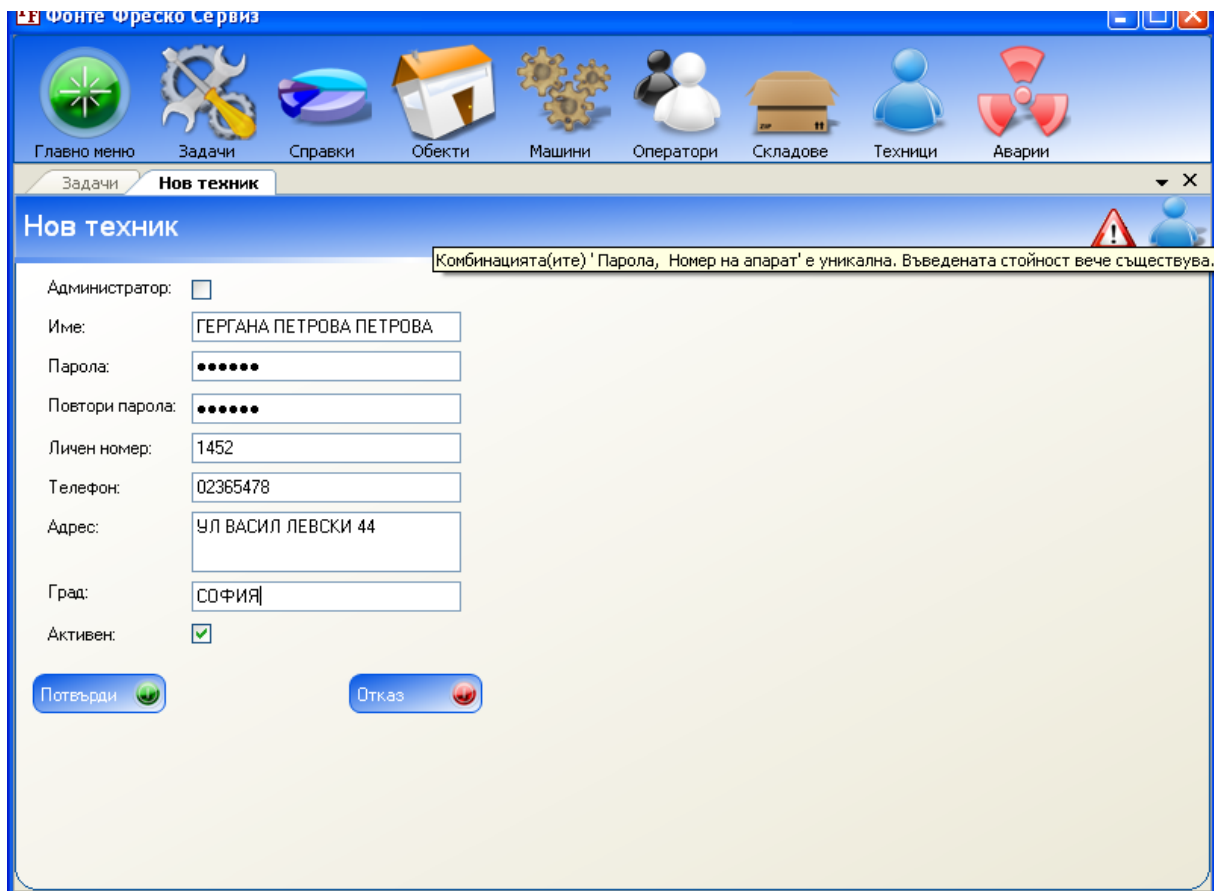


The screenshot shows a window titled "Интернет настройки" (Internet settings) with a close button in the top right corner. The main title is "Интернет настройки". Below the title are several settings fields: "Интернет адрес:" (Internet address) with a text box containing "http://www.fontefresco.bg/FonteFrescoWebServices/FonteFrescoService.asmx"; "Време за отговор:" (Response time) with a spinner box set to "60" and the unit "секунди" (seconds); "Свързване през прокси:" (Connect via proxy) with an unchecked checkbox; "Адрес на прокси:" (Proxy address) with an empty text box; and "Порт на прокси:" (Proxy port) with an empty text box. At the bottom left is a "Потвърди" (Confirm) button with a green arrow icon, and at the bottom right is an "Отказ" (Cancel) button with a red stop icon.

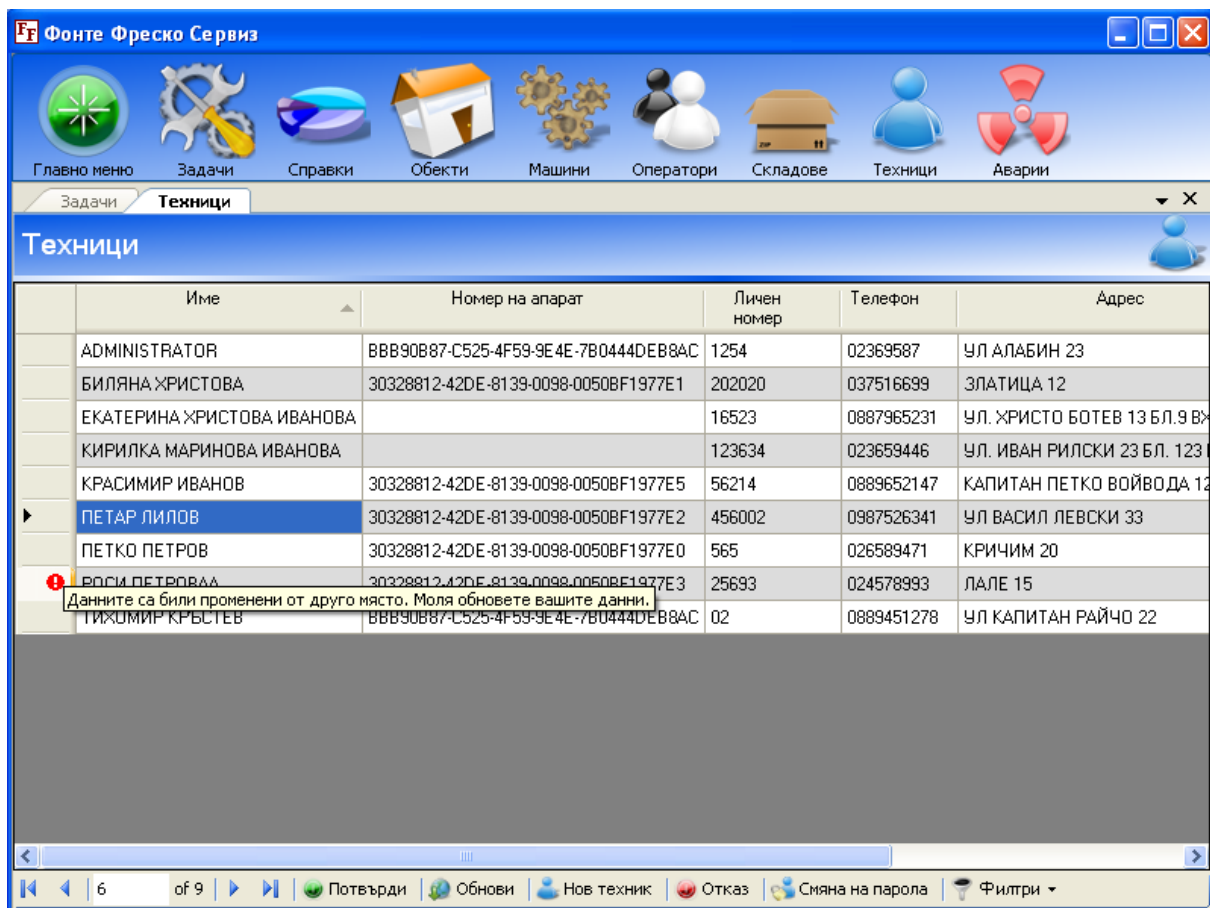
фиг. 61 Форма „Интернет настройки”



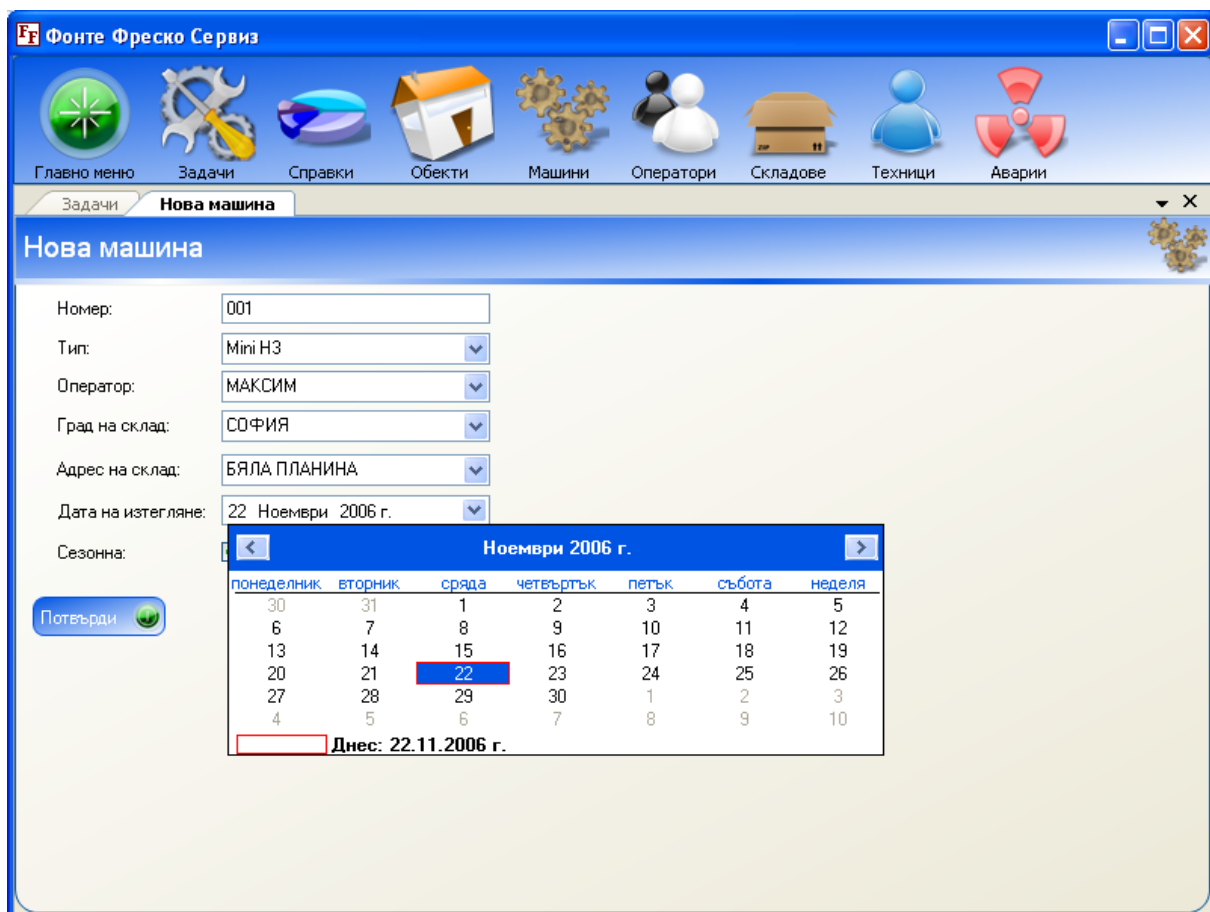
фиг. 62 Функционалността за drag-and-drop



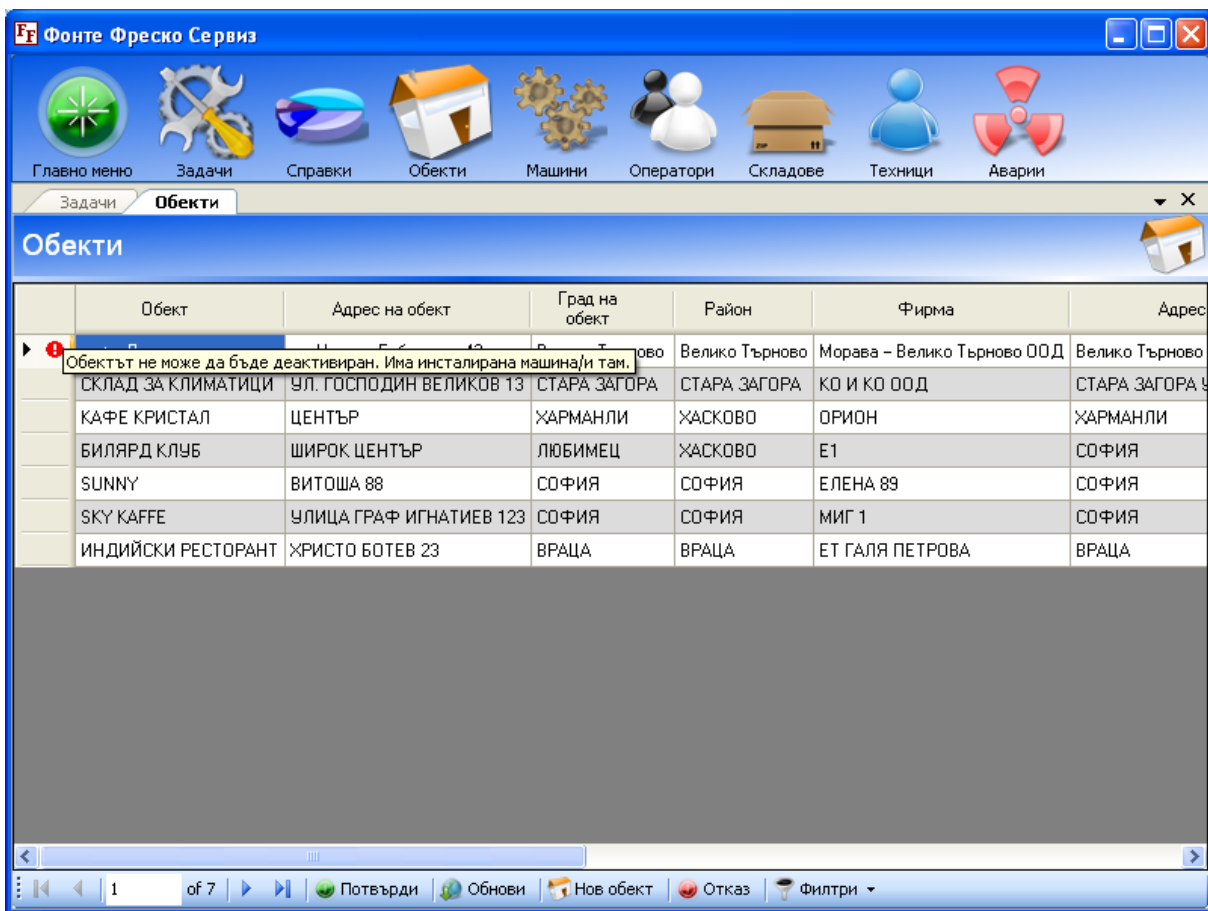
фиг. 63 Грешка при съхранение на данни за нов техник(Unique constraint)



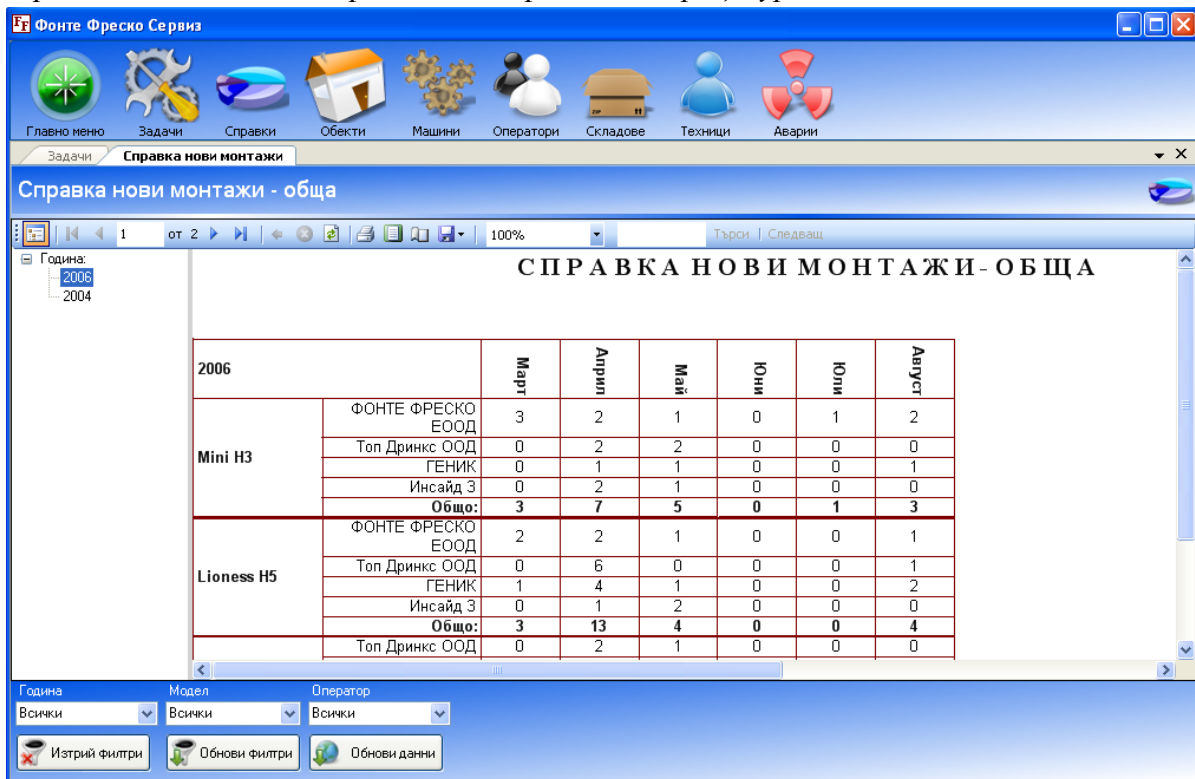
фиг. 64 Форма „Техници” – грешка при съхранение на промени



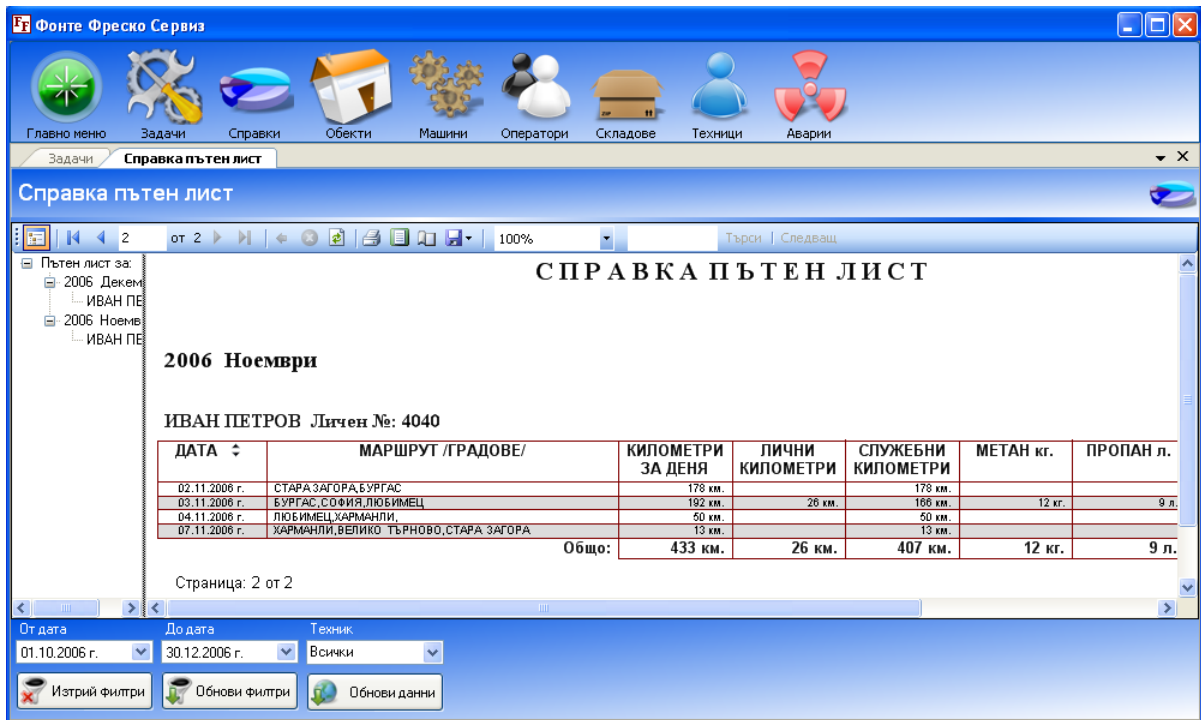
фиг. 65 Форма „Нова машина”



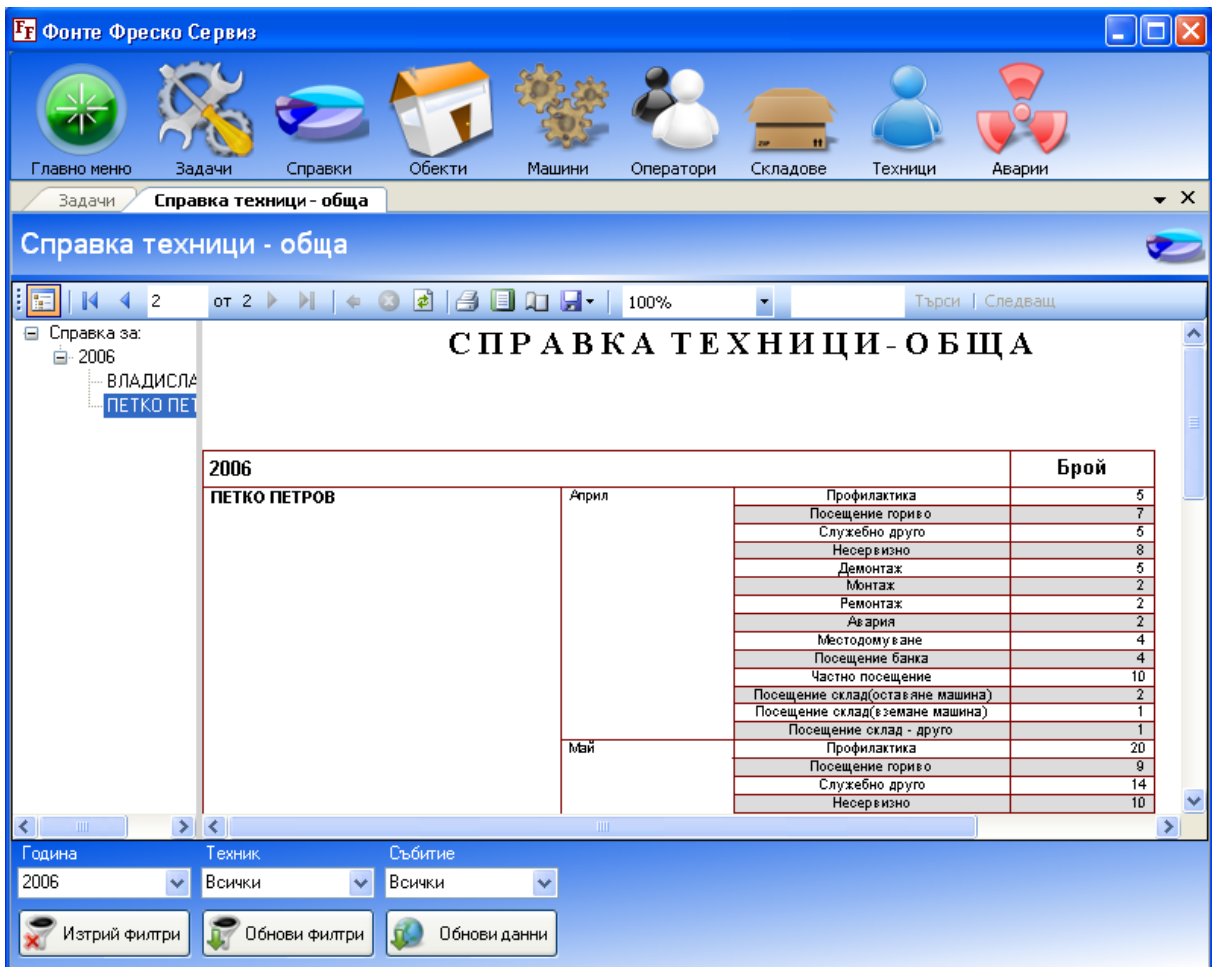
фиг. 66 Изключение хвърлено от съхранената процедура за обновяване на обектите



фиг. 67 Форма „Справка нови монтаж - обща“



фиг. 68 Форма „Справка пътен лист”



фиг. 69 Форма „Справка техници-обща”

Фонте Фреско Сервиз

Главное меню Задачи Справки Объекти Машини Оператори Складове Техници Аварии

Задачи **Справка техници - подробна**

Справка техници - подробна

1 от 2 100% Търси | Следващ

Техници
 ПЕТКО ПЕТРОВ
 ВЛАДИСЛАВ

СПРАВКА ЗА ТЕХНИЦИ - ПОДРОБНА

ПЕТКО ПЕТРОВ Личен №: 565

СЪБИТИЕ	КИЛОМЕТРАЖ	ИЗМИНАТИ КИЛОМЕТРИ	ДАТА	ВРЕМЕ ЗА РЕАКЦИЯ
Посещение гориво	7600	20	08.7.2006 г. 01:24	10 дни ,13часа ,18 ми
Частно посещение	7620	20	14.7.2006 г. 10:46	6 дни ,9часа ,22 ми
Местодомуване	7620	0	14.7.2006 г. 10:46	0 ми
Авария	7630	10	14.7.2006 г. 11:27	41 ми
Частно посещение	7635	5	14.7.2006 г. 13:20	1час ,53 ми
Профилактика	7660	25	19.7.2006 г. 13:39	5 дни ,19 ми
Авария	7667	7	19.7.2006 г. 14:24	45 ми
Частно посещение	667788	660121	24.7.2006 г. 17:06	5 дни ,2часа ,42 ми
Профилактика	667800	12	29.7.2006 г. 23:20	5 дни ,6часа ,14 ми
Монтаж	667900	100	31.7.2006 г. 01:38	1 ден ,2часа ,18 ми
Авария	667900	0	01.8.2006 г. 18:44	1 ден ,17часа ,8 ми
Посещение гориво	678900	11000	04.8.2006 г. 14:10	2 дни ,19часа ,26 ми
Частно посещение	678910	10	06.8.2006 г. 16:33	2 дни ,2часа ,23 ми
Служебно друго	678920	10	06.8.2006 г. 16:35	2 ми
Служебно друго	678933	13	06.8.2006 г. 16:37	2 ми
Посещение гориво	678940	7	06.8.2006 г. 16:40	3 ми

От дата: 01.7.2006 г. До дата: 30.11.2006 г. Техник: Всички Събитие: Всички

Изтрий филтри Обнови филтри Обнови данни

фиг. 70 Форма „Справка техници - подробна”

Фонте Фреско Сервиз

Главное меню Задачи Справки Объекти Машини Оператори Складове Техници Аварии

Задачи **Справка монтаж... и ремонтaжи**

Справка монтаж и ремонтaжи

3 от 6 100% Търси | Следващ

Тип инсталация:
 Монтаж на водопровод
 Монтаж на помпа
Монтаж на водопровод
 Монтаж на помпа
 Монтаж на водопровод
 Монтаж на доливане

СПРАВКА МОНТАЖИ И РЕМОНТАЖ

Монтаж на водопровод

ПЕЧАТ №	СЕРИЕН №	ОБЕКТ	ГРАД НА ОБЕКТ	АДРЕС НА ОБЕКТ	ДАТА	ОПЕРАТОР
565	465867	КАФЕ ДЖУРАСИК	ВЕЛИКО ТЪРНОВО	УЛ. НИКОЛА ГАБРОВСКИ 43	08.8.2006 г.	ФОНТЕ Ф
565	04	SKY KAFFE	СОФИЯ	УЛИЦА ГРАФ ИГНАТИЕВ 123	10.8.2006 г.	ФОНТЕ Ф
565	04	SKY KAFFE	СОФИЯ	УЛИЦА ГРАФ ИГНАТИЕВ 123	10.8.2006 г.	ФОНТЕ Ф

ОБЩО: 3

Страница: 3 от 6
 Изготвил: ADMINISTRATOR
 Дата: 23 Ноември 2006 14:41

От дата: 01.8.2006 г. До дата: 30.11.2006 г. Град на обект: Всички Обект: Всички Оператор: Всички Събитие: Всички Печат номер: Всички Серийен номер: Всички

Изтрий филтри Обнови филтри Обнови данни

фиг. 71 Форма „Справка монтаж и ремонтaжи”

Имплементация на формата за въвеждане на нов оператор

```
public partial class InsertOperatorForm : DockContent
{
    #region Constructors
    public InsertOperatorForm()
    {
        InitializeComponent();
    }
    #endregion

    #region Private Methods
    private void ShowWaitControl()
    {
        this.waitControl.ShowWaitControl();
        this.Refresh();
    }

    private void HideWaitControl()
    {
        if (this.InvokeRequired)
        {
            this.Invoke((MethodInvoker)delegate()
            { HideWaitControl(); });
            return;
        }

        if (this.waitControl != null)
        {
            this.waitControl.Visible = false;
        }
    }

    private void ClearFormControls()
    {
        foreach (Control control in this.gradientPanel.Controls)
        {
            if (control is TextBox)
            {
                ((TextBox)control).Clear();
            }
        }
        this.titleControl.Focus();
        this.validator.Clear();
        this.ShowErrorText("");
    }
    #endregion

    #region Event Handlers
    /// <summary>
    /// Hide form
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void cancelButton_Click(object sender, EventArgs e)
    {
        this.Close();
    }

    private void SaveData()
    {
        if (this.validator.Validate() == false)
            return;

        this.ShowWaitControl();
    }
    #endregion
}
```



```

this.operatorsDataSet = new OperatorsDataSet();
this.operatorsDataSet.Operators.AddOperatorsRow(
    1,
    this.nameTextBox.Text,
    this.molTextBox.Text,
    this.bulstatTextBox.Text,
    this.cityTextBox.Text,
    this.addressTextBox.Text,
    true,
    new byte[] { 1 });
TAdminWebService adminWebService =
    TAdminWebService.CreateAdminWebService();
ThreadPool.QueueUserWorkItem((WaitCallback) delegate (Object
                                                                    stateInfo)
    {
        try
        {
            OperatorsDataSet dataSet =
                adminWebService.UpdateOperators(
                    this.operatorsDataSet.GetChanges() as OperatorsDataSet);
            SaveDataCallback(dataSet);
        }
        catch (Exception ex)
        {
            HandleException(ex);
        }
        finally
        {
            this.HideWaitControl();
        }
    });
}

private void SaveDataCallback(OperatorsDataSet dataSet)
{
    if (this.InvokeRequired)
    {
        this.Invoke((DataDelegate<OperatorsDataSet>)
                    delegate (OperatorsDataSet data)
                    { SaveDataCallback(data); }, new object[] { dataSet });
        return;
    }

    if (dataSet.Operators.HasErrors)
    {
        this.ShowErrorText(FormUtils.ERROR_INSERTING_CAPTION);
    }
    else
    {
        MessageBox.Show(FormUtils.SUCCESSFUL_SAVING_DATA,
                        FormUtils.RECORD_FINISHED_CAPTION,
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Information);
        this.ClearFormControls();
    }
}

private void ShowErrorText(string errorText)
{
    this.internetErrorProvider.SetError(this.titleControl.ErrorLabel,
                                        errorText);
}

private void HandleException(Exception ex)
{
    if (this.InvokeRequired)
    {
        this.Invoke((MethodInvoker) delegate ()
                    { HandleException(ex); });
        return;
    }
}

```



```

FormUtils.LogException(ex);
if (ex is SoapException)
{
    If
    (ex.Message.Contains(FormUtils.UNIQUE_CONSTRAINT_OPERATORS_NAME))
    {
        string message = FormUtils.GetUniqueConstraintMessage(
            FormUtils.UNIQUE_CONSTRAINT_OPERATORS_NAME,
            this.operatorsDataSet.Operators);
        this.ShowErrorText(message);
        return;
    }

    if (ex.Message.Contains(FormUtils.SQL_ERROR_EMPTY_OR_NULL_DATA))
    {
        this.ShowErrorText(FormUtils.ERROR_EMPTY_OR_WRONG_DATA_TEXT);
        return;
    }

    if (ex.Message.Contains(FormUtils.ERROR_AUTHENTICATION))
    {
        MessageBox.Show(this, FormUtils.ERROR_DENY_USER_ACCESS_TEXT,
            FormUtils.ERROR_GENERAL_TEXT,
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        MainForm parent = (this.MdiParent as MainForm);
        if (parent != null)
        {
            parent.Terminate = true;
            parent.Close();
        }
    }

    this.ShowErrorText(FormUtils.ERROR_SERVER_PROCESSING);
    return;
}

if (ex is WebException)
{
    this.ShowErrorText(FormUtils.ERROR_INTERNET_CONNECTION_TEXT);
    return;
}

this.ShowErrorText(FormUtils.ERROR_GENERAL_TEXT);
}

/// <summary>
/// Create new row with entered data in Personnel table.
/// And call web method do update Personnel table in DataBase
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void okButton_Click(object sender, EventArgs e)
{
    SaveData();
}

private void InsertOperatorForm_FormClosing(object sender,
                                           FormClosingEventArgs e)
{
    e.Cancel = true;
    if (this.waitControl.Visible == false)
    {
        this.ClearFormControls();
        this.operatorsDataSet.Dispose();
        this.Hide();
        this.DockPanel = null;
    }
}
}
#endregion

```