**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Саратовский государственный технический университет**

**имени Гагарина Ю.А.»**

Факультет Международный факультет прикладных информационных технологий

Специальность Информационные системы и технологии

Кафедра Прикладные информационные технологии

**ДИПЛОМНАЯ РАБОТА**

«Разработка программного обеспечения для учета серверов, находящихся в промышленной эксплуатации, и удаленного развертывания приложений»

Выполнил студент группы ИСТ-51

Бороздюхин А.Ю.

Руководитель работы к т.н., доцент

каф. ПИТ Большаков А.А.

Допущен к защите

Протокол № 25 от 09 июня 2014 г.

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Долинина О.Н.

Саратов 2014

**РЕФЕРАТ**

Отчет 84 страницы, 3 части, 36 рисунков, 18 таблиц, 31 источник, 3 приложения. Цель работы – повышение эффективности работы специалистов по разработке программного обеспечения ЗАО "ТНДМ Софт" за счет создания ПО для централизованного хранения информации о продуктивных серверах и удаленного развертывания приложений. Объектом автоматизации является ЗАО "ТНДМ Софт" – компания, предоставляющая услуги в области ИТ-аутсорсинга. В процессе выполнения работы было проведено детальное сравнение существующих систем и установлен факт несоответствия требованиям заказчика. Выбор программного обеспечения определялся исходя из возможностей языка программирования и наличия соответствующих лицензий. Для создания информационной системы была проведена работа по исследованию предметной области, выявлена структура и особенности объекта автоматизации. По результатам исследования была спроектирована база данных, отображающая особенности предметной области, и разработано клиентское приложение, предоставляющее интерфейс для работы с базой данных. В результате проведенных работ был получен полнофункциональный программный продукт, отвечающий всем предъявленным требованиям. Пояснительная записка выполнена в текстовом редакторе Microsoft Office Word 2013. Файл, содержащий пояснительную записку, имеет название: "Диплом Бороздюхин.doc"

Оглавление

[ВВЕДЕНИЕ 5](#_Toc389782045)

[СПИСОК СОКРАЩЕНИЙ 7](#_Toc389782046)

[1. ОПИСАНИЕ ОБЪЕКТА АВТОМАТИЗАЦИИ 8](#_Toc389782047)

[1.1. Текущая модель разработки и обновления ПО в компании «ТНДМ Софт» 8](#_Toc389782048)

[1.2. Характеристики объекта автоматизации 9](#_Toc389782049)

[1.3. Постановка задачи 9](#_Toc389782050)

[1.4. Целевая модель процесса обновления 11](#_Toc389782051)

[1.5. Преимущества внедрения системы 11](#_Toc389782052)

[2. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ 12](#_Toc389782053)

[2.1. System Center Configuration Manager 12](#_Toc389782054)

[2.2. WebDeploy + Visual Studio 15](#_Toc389782055)

[2.3. Power Shell / Язык оболочки 18](#_Toc389782056)

[3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ DOUIN 21](#_Toc389782057)

[3.1. Функциональные возможности 21](#_Toc389782058)

[3.2. Входные и выходные данные 22](#_Toc389782059)

[3.3. Схема работы приложения 23](#_Toc389782060)

[3.4. Архитектура системы 24](#_Toc389782061)

[3.4.1. Уровень доступа к данным 25](#_Toc389782062)

[3.4.2. Слой бизнес логики приложения 27](#_Toc389782063)

[3.4.2.1. Модуль компиляции 27](#_Toc389782064)

[3.4.2.2. Модуль развертывания приложений 28](#_Toc389782065)

[3.4.2.3. Модуль формирования отчетов 30](#_Toc389782066)

[3.5. Алгоритм обновления приложений 30](#_Toc389782067)

[3.6. Выбор технологий для разработки системы 31](#_Toc389782068)

[3.6.1. Выбор платформы 31](#_Toc389782069)

[3.6.2. Выбор языка программирования 36](#_Toc389782070)

[3.6.3. Выбор технологии доступа к данным 38](#_Toc389782071)

[3.6.4. Выбор СУБД 41](#_Toc389782072)

[3.6.5. Выбор технологии разработки веб-приложения 42](#_Toc389782073)

[3.6.6. Выбор технологий построения пользовательского интерфейса 48](#_Toc389782074)

[3.6.6.1. jQuery 48](#_Toc389782075)

[3.6.6.2. Twitter Bootstrap 49](#_Toc389782076)

[3.6.7. Выбор технологии построения отчетов 50](#_Toc389782077)

[3.7. Описание классов системы 52](#_Toc389782078)

[3.8. Проектирование базы данных системы 56](#_Toc389782079)

[3.9. Описание графического интерфейса системы 64](#_Toc389782080)

[3.9.1 Добавление нового сервера 66](#_Toc389782081)

[3.9.2. Конфигурирование объектов оповещения 68](#_Toc389782082)

[3.9.3. Развертывание приложения 71](#_Toc389782083)

[3.9.4. Работа с отчетами 72](#_Toc389782084)

[3.9.4.1. Разработка отчетов 72](#_Toc389782085)

[3.9.4.2. Добавление отчетов в систему 73](#_Toc389782086)

[3.9.4.3. Формирование отчетов 74](#_Toc389782087)

[3.10. Аппаратное и программное обеспечение, необходимое для внедрения разработанной системы 77](#_Toc389782088)

[ЗАКЛЮЧЕНИЕ 79](#_Toc389782089)

[СПИСОК ЛИТЕРАТУРЫ 80](#_Toc389782090)

[ПРИЛОЖЕНИЕ 1. Реализация Базового репозитория. 83](#_Toc389782091)

[ПРИЛОЖЕНИЕ 2. Реализация CRUD Контроллера 84](#_Toc389782092)

[ПРИЛОЖЕНИЕ 3. Отсканированная страница Акта о внедрении ПО 85](#_Toc389782093)

# ВВЕДЕНИЕ

В настоящее время существует огромное количество компаний, занимающихся предоставлением услуг в сфере ИТ – аутсорсинга. С ростом количества заказчиков, остро встает вопрос автоматизации различных стадий разработки программного обеспечения, в частности – процесса обновления ПО на серверах, так как при большом количестве клиентов, как правило, это занимает достаточно большое количество времени. Современный рынок программного обеспечения разнообразен, в том числе, существует множество продуктов, способных автоматизировать процесс обновления версий разрабатываемого программного обеспечения на удаленных серверах, но зачастую, по ряду причину, не всегда удается использовать готовое решение. К таким причинам можно отнести:

* сложность внедрения в инфраструктуру предприятия;
* дороговизна внедрения;
* сложность поддержки.

Целью дипломной работы является разработка программного продукта, который позволит:

* автоматизировать процесс обновления версий разрабатываемого ПО на продуктивных серверах заказчиков;
* автоматизировать учет серверов и версий установленных приложений;
* автоматизировать сбор статистики по обновлениям ПО и выдачу отчетов.

Для достижения целей работы необходимо выполнить следующие задачи:

* проведение анализа объекта автоматизации;
* составление списка требований к разрабатываемой системе;
* проведение анализа рынка аналогичных систем;
* разработка архитектуры системы;
* разработка алгоритмов работы системы;
* разработка пользовательского интерфейса системы;
* разработка базы данных системы;
* разработка подсистемы построения отчетов;
* разработка подсистемы развертывания приложений;
* разработка подсистемы компиляции приложений;
* создание конечного варианта системы;
* проведение тестовой эксплуатации.

Актуальность работы обоснована наличием заказа от предприятия ЗАО «ТНДМ Софт», предоставляющего услуги в области ИТ-аутсорсинга.

# СПИСОК СОКРАЩЕНИЙ

|  |  |
| --- | --- |
| ***Сокращение*** | ***Расшифровка*** |
| ПО | Программное обеспечение |
| SCCM | System Center Configuration Manager |
| RDP | Remote Desktop Protocol |
| IIS | Internet Information Services |
| GAC | Global Assemblies Cache |
| CRUD | Create Read Update Delete |
| СУБД | Система управления базами данных |
| EF | Entity Framework |
| CLR | Common Language Runtime |
| CTS | Common Type System |
| CLS | Common Language Specification |
| SSRS | Sql Server Reporting Services |
| ORM | Object-relational mapping |
| MVC | Model View Controller |

# **1. ОПИСАНИЕ ОБЪЕКТА АВТОМАТИЗАЦИИ**

## 1.1. Текущая модель разработки и обновления ПО в компании «ТНДМ Софт»

Компания «ТНДМ Софт» занимается предоставлением аутсорсинговых услуг в сфере информационных технологий, в частности, услуг по внедрению информационных систем. Вместе с внедрением ПО практически всегда покупается техническая поддержка, в рамках которой постоянно обновляются версии разработанного программного продукта.

Разработка программного обеспечения – итеративный процесс, и итерации могут быть сколь угодно длинными. Самая минимальная – это выпуск нового обновления, то есть пакета программного средства, который призван исправить определенные проблемы в приложении. Следующая по продолжительности итерация – это выпуск новой версии продукта. Внутри компании «ТНДМ Софт» среднее время выпуска версии – 2 недели. Выпущенный пакет тестируется, и если он оказывается стабильным и удовлетворяющим текущим заявкам пользователей, переводится в статус внутренней версии ПО. Тестирование внутреннего версии очень ответственный момент, так как результатом тестирования будет приложение, которое устанавливается на серверах заказчиков. Если в процессе функционирования системы у заказчика выявляют какие-либо критичные ошибки, мешающие работе в системой, выпускается пакет обновления, который так же публикуется на серверах заказчиков.

Процесс обновления программного продукта на серверах приложений заказчиков условно можно разделить на несколько шагов:

1. Поиск информации об удаленном сервере заказчика, которая обычно хранится в виде простого текстового файла.
2. Сборка текущей стабильной версии приложения в инсталляционный пакет.
3. Закачивание данного пакета на удаленный сервер.
4. Разворачивание инсталляционного пакета на IIS.
5. Тестирование развернутого пакета.

Эти шаги повторяются для всех серверов, на которых установлено программное обеспечение компании.

## 1.2. Характеристики объекта автоматизации

«ТНДМ Софт» имеет три филиала в трех городах России. Головной офис компании находится в Москве, Центр разработок в Саратове и Тюмени. Важные числовые характеристики предприятия приведены ниже:

* общая численность сотрудников компании – 68 человек;
* количество пользователей решений компаний – около 10000 человек;
* количество потенциальных пользователей разработанной системы – 20-30 человек (разработчики и менеджеры компании);
* на каждого сотрудника приходится по 1 компьютеру;
* среднее время выпуска нового пакета обновления– 2 недели;
* общее количество внедренных решений – более 100;
* среднее время обновления одного приложения на удаленном сервере – 20 минут;
* Количество серверов заказчиков, обслуживаемых компанией – несколько десятков.

Таким образом, специалисту отдела разработки необходимо потратить целый рабочий день на обновление ПО всех продуктивных серверов. При использовании разработанного приложения, это время будет сокращено до нескольких часов, а при правильном конфигурировании системы, время обновления может сократиться еще больше.

## 1.3. Постановка задачи

С ростом количества проектов, остро был поднят вопрос автоматизации процесса обновления программного обеспечения на серверах заказчиков, так как автоматизация данного процесса сможет повысить эффективность работы отдела разработки программного обеспечения, а, следовательно, и уменьшить издержки компании. Существующие программные средства не могут покрыть все требования, предъявляемые к системе обновления приложений на серверах заказчика. Возникает необходимость разработки собственного программного обеспечения. Разрабатываемую систему принято было назвать DOUIN (Deploy Or Update It Now). Заказчиком были выдвинуты следующие требования к разрабатываемой системе:

* возможность хранения информации о продуктивных серверах;
* возможность выкачивания исходных файлов из репозитория системы контроля версии для каждого из приложения сервера;
* возможность компиляции полученных исходных файлов;
* возможность развертывания скомпилированного приложения на удаленном сервере;
* наличие системы построения отчетов на основе SQL Server Reporting Services;
* наличие эффективной системы поиска объектов в системе;
* высокая степень масштабируемости системы;
* минимальные финансовые затраты на внедрение системы;
* минимальные изменения текущей инфраструктуры предприятия;
* минимальное количество оборудования и занимаемого системой пространства.

На основе данных требований был проведен анализ существующих решений, а также они были учтены при проектировании, разработке и выборе инструментария разработки системы.

## 1.4. Целевая модель процесса обновления

В настоящее время информация о продуктивных серверах хранится в нескольких текстовых файлах и подключение для развертывания приложений происходит с использованием различных технологий (к примеру, RDP). После внедрения системы DOUIN информация о серверах будет храниться централизованно в базе данных.

Доступ ко всем объектам системы предоставляется через веб – интерфейс. Все внутренние механизмы инкапсулированы, и пользователь никак не может повлиять на логику развертывания. Исключение составляют лишь те части программного обеспечения, которые поддаются конфигурированию (более подробно описано далее).

Так же пользователь имеет возможность просмотреть информацию обо всех объектах системы, в частности, строить отчеты, основанные на исторических данных. Это возможно потому, что практически все операции, которые делает пользователь в системе, логируются в базу данных.

Любой сформированный отчет может быть экспортирован в любой из трех форматах: PDF, Excel, Word. Так же сформированный отчет может быть отправлен по электронной почте любому количеству зарегистрированных в системе объектах пользователей.

## 1.5. Преимущества внедрения системы

Во время внедрения системы DOUIN в эксплуатацию, в нее будет загружена информация о всех серверах заказчиков, которые обслуживаются компанией. Это обеспечит надежное хранение данной информации с удобным доступом к ней. Использование разработанной системы поиска информации в системе позволит пользователям быстро находить нужные объекты. Кроме того, вся информация надежно защищена и доступна только пользователям домена, внутри которого будет работать данная система.

Таким образом, внедрение системы DOUIN позволит управлять большим количество серверов простыми манипуляциями в интерфейсе приложения. Экономическая выгода от внедрения подобной системы очевидна: использование DOUIN сократит время обновления приложений на продуктивных серверах, что сделает труд разработчиков более продуктивным, и, по приблизительным оценкам, обновление одного приложения сократится на 80-90%. Скорость обновления зависит от нескольких факторов: скорости интернет – канала, объема исходных файлов, мощности процессора сервера приложений, на котором будет работать данная система, то есть время совершения операции никаким образом не зависит от пользователя.

# 2. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

Ниже приведен сравнительный анализ технологий, наиболее популярных среди специалистов для решения задач централизованного удаленного развертывания.

## 2.1. System Center Configuration Manager

**Описание технологии**

System Center Configuration Manager – продукт для управления ИТ –инфраструктурой предприятия на основе Microsoft Windows. С помощью данного инструмента возможно обновления и развертывание ПО, операционных систем, ведение учета программного и аппаратного обеспечения.

SCCM представляет собой мощный инструмент для решения разнообразных задач, в том числе контроль всей ИТ – инфраструктуры предприятия. Управление клиентами (устройства, мобильные устройства, рабочие станции) осуществляется через клиентское программное обеспечение SCCM. Под управлением подразумевается настройка различных параметров клиентов, установка программного обеспечения, генерация отчетов о текущем состоянии программного и аппаратного обеспечения устройств.

**Возможности технологии**

SCCM предусматривает несколько вариантов установки приложений на клиентах: обновления, ручная установка, принудительная установка. Для более удобного централизованного управления клиентскими устройствами в SCCM реализована возможность объединение несколько устройств в коллекцию по некоторым критериям. К примеру, все мобильные устройства можно выделить в коллекцию «Мобильные устройства» и к данной коллекции применять какие-либо управляющие действия, например, обновление мобильного приложения на всех устройствах в данной коллекции. Таким образом можно создавать различные группы устройств исходя из бизнес правил.

В SCCM существуют два типа коллекций:

* Встроенные коллекции
* Пользовательские коллекции

Встроенные коллекции:

1. **Все группы пользователей.** Содержит список групп пользователей, который формируется через Active Directory
2. **Все пользователи.** Содержит список пользователей, который формируется через Active Directory
3. **Все пользователи и группы пользователей.** Содержит список всех пользователей и групп пользователей
4. **Все клиенты сервера и рабочего стола.** Содержит список всех устройств, которые являются клиентами SCCM. То есть список тех устройств, на которых установлено клиентское программное обеспечение SCCM.
5. **Все мобильные устройства.** Содержит список мобильных устройств, которые включены в управление SCCM.
6. **Все системы.** Содержит список всех устройств, в том числе устройств, которые не являются клиентами SSCM.
7. **Все неизвестные компьютеры.** Содержит список всех устройств, которые не сконфигурированы как клиент SCCM.

Формирование пользовательских коллекций происходит по нескольким правилам:

1. **Правило направления.** По данному правилу можно получить прямой доступ к устройству для добавления его в коллекцию. Этот тип коллекций имеет некоторые административные издержки, так как управление данным типом коллекции осуществляется вручную.
2. **Правила запроса.** Это правило позволяет добавлять клиенты в коллекцию динамически. Например, можно создать коллекцию пользователей, являющихся сотрудниками конкретного подразделения предприятия. Таким образом пользователи автоматически будут попадать в данную коллекцию.

Примеры подобных запросов:

* Все пользователи подразделения;
* Все рабочие станции под управление Windows 7;
* Все рабочие станции, имеющие менее 15 ГБ свободного места на жестком диске.

1. **Включение коллекций.** Это правило позволяет включать с коллекцию элементы из другой коллекции по каким-то заранее заданным правилам или по расписанию.
2. **Исключение коллекций.** Это правило позволяет исключать из коллекции элементы по каким-то критериям или по расписанию.

SCCM безусловно очень мощный профессиональный инструмент с большими возможностями, который можно настроить для выполнения поставленной задачи, но чтобы работать с данным продуктом, нужны специально обученные специалисты. Так же следует отметить дороговизну внедрения данного решения на предприятии.

## 2.2. WebDeploy + Visual Studio

**Описание технологии** Web Deploy [1] широко используется в процессах разработки программного обеспечения на платформе .NET. MS Web Deploy – это технология, предназначенная для упрощения процессов развертывания веб – сайтов и приложений на сервере IIS. С помощью данной технологии возможно развертывание приложений, регистрация компонентов в GAC и многое другое. Web Deploy позволяет эффективно синхронизировать приложения на ферме веб – серверов. Приложение так же упрощает процесс конфигурирования приложений на фермах, в том числе, возможно в процессе конфигурации регистрировать компоненты в GAC. Важным моментом в использовании данного инструмента является то, что возможно сконфигурировать систему таким образом, что процесс развертывания сможет происходить из-под специально настроенной учетной записи, у которой не будет административных привилегий. Установку Web Deploy можно осуществить, например, через инструмент Web Platform Installer, который скачает и установит на нужной машине последнюю версию приложения. На рисунке 2.1 показан процесс настройки установленного Web Deploy для конкретного сайта IIS [2].

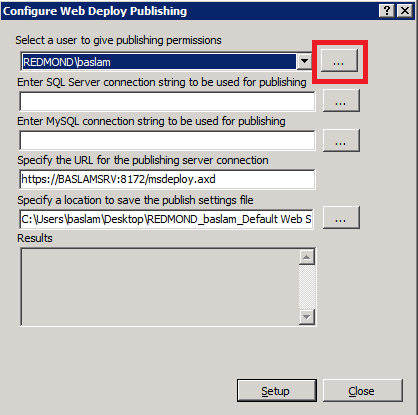


Рис. 2.1. Конфигурирование Web Deploy

Для конкретного сайта задаются параметры, необходимые для развертывания приложения. В том числе, задается учетная запись, из-под которой будет происходить развертывание (Рисунок 2.2).

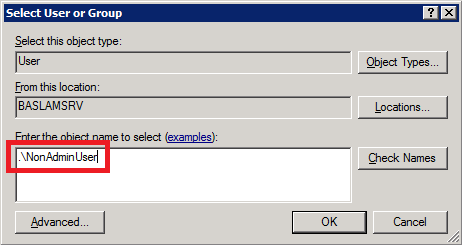


Рис. 2.2. Учетная запись для Web Deploy

Visual Studio – это среда разработки приложений, поддерживающая шаблоны создания Windows приложений, веб – приложений, библиотек классов и т.д. Visual Studio включает в себя редактор исходного кода с поддержкой функции IntelliSense и инструменты для простейшего рефакторинга кода. Интерфейс Visual Studio позволяет разработчику взаимодействовать с базами данных, различными редакторами классов, системой контроля версий и проч. Для использования данного инструмента в командной разработке, существует поддержка клиента/сервера Team Foundation Server. На рисунке 2.3 представлен интерфейс среды разработки.

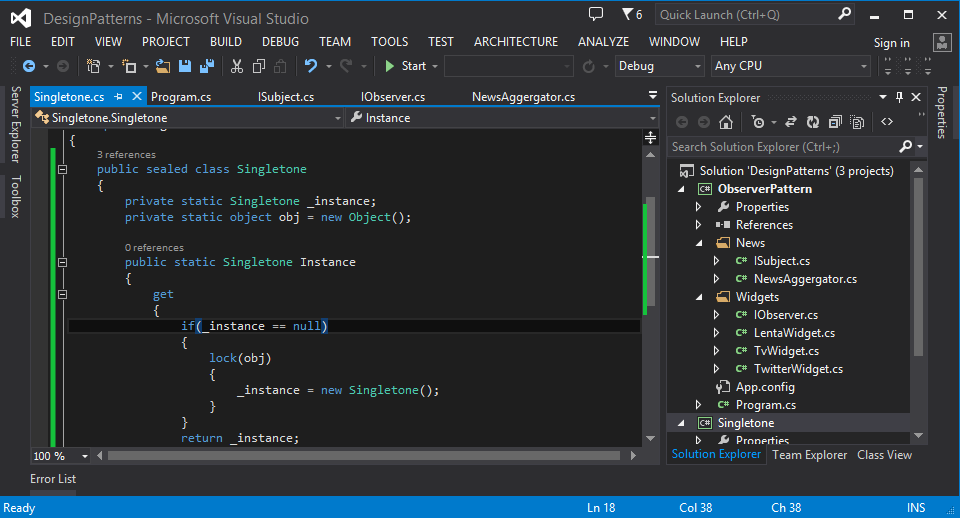


Рис. 2.3. Среда разработки Visual Studio 2013

Web Deploy является встраиваемым компонентом в среду Visual Studio, то есть это значит, что взаимодействие с некоторыми функциями Web Deploy возможно через интерфейс Visual Studio. В том числе - развертывание веб приложений через Visual Studio. Но чтобы начать пользоваться Web Deploy через Visual Studio, необходимо сначала настроить профиль публикации. Окно настройки данного профиля представлено на рисунке 2.4.

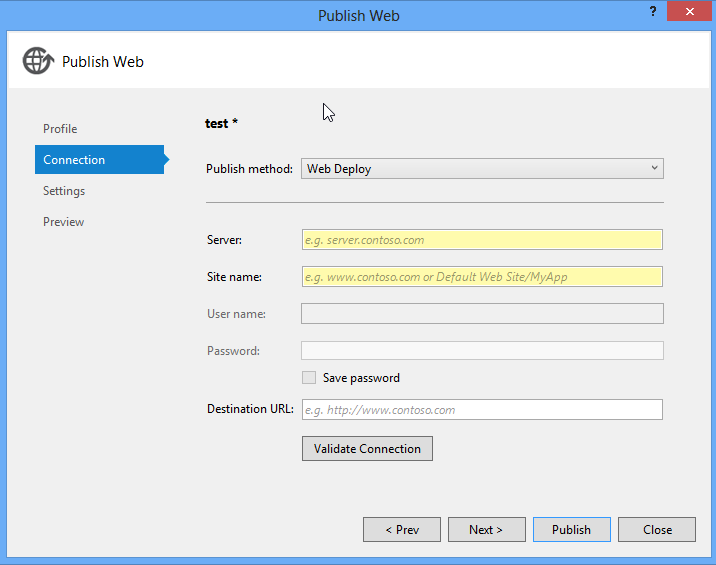


Рис. 2.4. Настройка профиля публикации

После настройки необходимых параметров, можно приступать к публикации приложения. Таким образом обеспечивается удаленное развёртывание приложений, но не решается задача централизованного хранения информации о продуктивных серверах. Так же отсутствует возможность хранения исторической информации, которая нужна для построения отчетов и ведения статистики, а также, используя данный подход, невозможно обновить несколько удаленных серверов, что, собственно является первоочередной задачей.

## 2.3. Power Shell / Язык оболочки

**Описание технологии** Windows Power Shell – средство автоматизации от Microsoft, которое включает в себя оболочку с интерфейсом командной строки и интегрированный язык сценариев. Windows Poser Shell построен на базе .NET Framework и интегрирован с ним. Как дополнение, Power Shell предоставляет более удобный интерфейс взаимодействия с объектами WMI и COM. Административные задачи решаются с помощью командлетов, в которых можно обращаться к различным утилитам и использовать средства командной строки Windows. Windows Power Shell так же реализует механизм встраивания, благодаря которому компоненты Power Shell могут быть встроены в другие приложения. На рисунке 2.5 представлен интерфейс оболочки Power Shell.

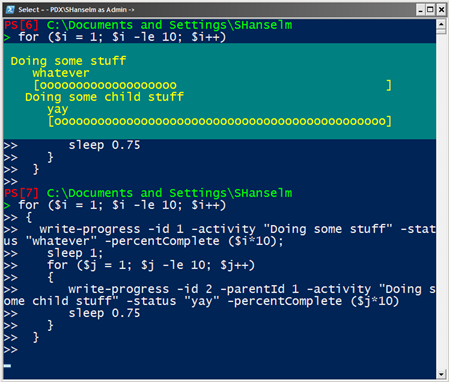


Рис. 2.5. Оболочка Power Shell

Язык оболочки – это встроенный в Windows скриптовый язык, использование которого возможно через командную строку операционной системы. На рисунке 2.6 представлен пример скрипта для командной строки Windows.

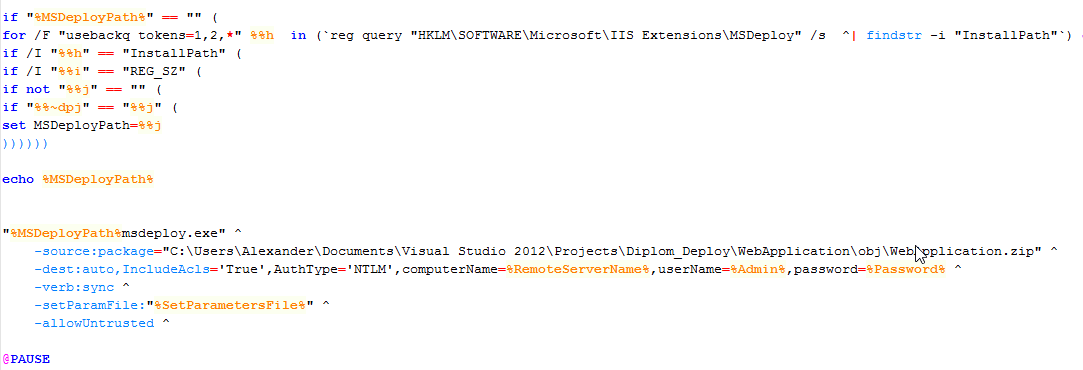


Рис. 2.6. Скрипт командной строки

Вышеописанные инструменты предоставляют широкие возможности по административному управлению операционной системы. В том числе, данные средства способны решить поставленную задачу по централизованному хранению информации о серверах и удаленному развертыванию приложений. Но, как известно, поддержка большого количества скриптов зачастую очень неудобна, так как в большом количестве файлов очень легко допустить ошибку, что просмотреть, забыть. Так же затрудняются вопросы масштабирования данной системы.

**Сравнительный анализ существующих решений**

В таблице 1 представлена сравнительная характеристика существующих решений.

Таблица 1. Сравнительная характеристика существующих решений

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Характеристика | SCCM | VS + WDeploy | Скриптовые языки | DOUIN |
| Централизованное  Хранение информации о серверах | + | - | + | + |
| Удаленное развертывание | + | + | + | + |
| Не требует специальной подготовки пользователей | - | + | - | + |
| Невысокая стоимость внедрения | - | + | + | + |
| Построение отчетов | + | - | - | + |
| Легкость масштабирования | + | - | - | + |
| Оповещение пользователей о действиях в системе | + | - | - | + |
| Настройка параметров системы без вмешательства в исходный код | + | - | - | + |

После изучения программных средств, обладающих аналогичным функционалом, было принято решение о необходимости разработки собственного решения, так как существующее программное обеспечение либо не полностью покрывает требования, либо внедрение решения затруднено с учетом существующей инфраструктуры и специфики работы предприятия.

# 3. ПРАКТИЧЕСКАЯ РЕАЛИЗАЦИЯ СИСТЕМЫ DOUIN

## 3.1. Функциональные возможности

В системе DOUIN реализован следующий функционал:

1. Хранение информации о продуктивных серверах.

В системе должна быть возможность добавления, удаления, редактирования информации об удаленных серверах, и объектах, которые являются частью данного сервера (сайты и виртуальные каталоги IIS)

1. Удаленное обновление приложений

Через интерфейс системы должна быть возможность обновления приложения на удаленном сервере с предварительным выкачиванием исходных модулей из репозитория контроля версии и компиляцией исходных файлов.

1. Построение отчетов

В системе должна быть возможность построения отчетов для более удобного представления информации об объектах системы конечным пользователям.

1. Просмотр статистической информации

В системе должна быть реализована возможность сохранения информации в базу данных обо всех операциях пользователей в системе.

## 3.2. Входные и выходные данные

Вся работа системы DOUIN основана на входных данных, которые в систему вносит пользователь. **Входные данные**

* Адрес удаленного сервера
* Порт удаленного сервера
* Учетные данные для подключения к серверу
  + Параметры приложений сервера
    - Название приложения
    - XML Кастомизация
    - Адрес репозитория управления версиями
    - Учетные данные для подключения к репозиторию управления версиями
    - Ревизия, которую нужно выкачать из репозитория
      * Объекты оповещения о развертывании приложения
        + Фамилия, имя, отчество
        + Адрес электронной почты, на которую отправится оповещение
        + Описание

**Результаты**

* Обновленное приложение на удаленном сервере
* Отчет об операции обновления
* Скомпилированный пакет приложения
* Отчеты, запрошенные пользователями

На рисунке 3.1 представлены входные/выходные данные в графическом виде.

**Параметры сервера**

**Параметры приложения**

**Параметры объектов оповещения**

**Обновленное приложение**

**Скомпилированный пакет**

**Отчетность**

**DOUIN**

Рис. 3.1. Входные/Выходные данные системы

## 3.3. Схема работы приложения

На рисунке 3.2 представлена схема работы приложения. Самое первое действие приложения, когда пользователь отправляет запрос на обновление какого-либо объекта – поиск в базе данных информации о сервере, на котором нужно обновить приложение, и информация о самом обновляемом приложении. На основе данных о репозитории системы контроля версий, система выкачивает исходные файлы. Если компиляция данного приложения происходит первый раз, то система предложит выбрать главный файл для компиляции, так как решение может включать в себя несколько проектов, а для компиляции нужен главный. В дальнейшим, если будет обновляться данное приложение, пользователю больше не будет предоставляться возможность выбора файла для компиляции, так как после первой компиляции данная информация сохранена в системе. Если компиляция прошла успешно, на основе данных удаленного сервера, приложение отправляет скомпилированный zip пакет удаленной службе, которая разворачивает данный zip пакет на удаленном IIS, предварительно применив правила кастомизации. После всех этих действий, пользователь получает информацию о ходе развертывания в текстовом виде, которая может быть сохранена на диск. Так же систему можно настроить таким образом, что при обновлении приложения на удаленном сервере, происходило email - информирование об этом всех заинтересованных лиц. Этот компонент настраивается в параметрах приложения сервера.



Sql Server

SVN Репозиторий

Zip spackage

Рис. 3.2. Схема работы приложения

## 3.4. Архитектура системы

Эффективность работы системы во многом определяется ее архитектурой. Проанализировав требования к системе, была выбрана архитектура, которая позволяет системе полностью соответствовать предъявляемым требованиям и максимально эффективно использовать имеющуюся инфраструктуру предприятия, при этом минимизировав загруженность канала связи с сервером.

Системы построена с учетом требований классической трехслойной архитектуры [3] и состоит из следующих компонентов:

* Слой доступа к данным
  + - * Классы - репозитории
      * DataManager
* Слой бизнес – логики
  + Модуль компиляции приложения
  + Модуль развертывания приложения
  + Модуль построения отчетов
* Интерфейс пользователя

На рисунке 3.3 представлена схема архитектуры системы.

* Entity Framework
* Классы – репозитории
* Data Manager
* Модуль компиляции приложений
* Модуль развертывания приложений
* Модуль построения отчетов

Интерфейс пользователя

БД

SSRS

Слой доступа к данным

Слой бизнес логики

Рис. 3.3. Архитектура системы

Рассмотрим архитектуру системы более подробно.

### 3.4.1. Уровень доступа к данным

В качестве технологии доступа к данным используется ORM Entity Framework [4]. На рисунке 3.4 представлена декомпозиция слоя доступа к

Хранилище данных

данным.

Фасад

Data Manager

IDao<T>

Объектная модель EF

Хранилище данных

Рис. 3.4. Декомпозиция слоя доступа к данным

Описание основных компонентов слоя доступа к данным:

* **Интерфейс IDao<T>**

Ключевым элементом в представленной архитектуре является обобщенный интерфейс Idao<T>. Данный интерфейс описывает CRUD операции над всеми типами сущностей. Логика манипулирования данными сущностей описывается в классах – репозиториях, каждый из который наследует Idao<T>. Доступ пользователя к классам – репозиториям осуществляется через класс – обертку DataManager.

* **Класс DataManager**

В данном классе реализован паттерн «Ленивая инициализация», который предоставляет удобный интерфейс взаимодействия с объектами репозитория. Использование данного класса сокращает общий объем программного кода, тем самым ускоряя время разработки.

* **Класс Фасад**

Из-за того, что CRUD операции для основных сущностей приложения являются комплексными (помимо работы с основными данными в базу данных так же записывается информация о некоторых действиях пользователей, а именно, информация о создании сущностей, информация об обновлении сущностей, удалении сущностей, об операциях развертывание и компиляции приложений) появилась необходимость применения паттерна «Фасад», который способен инкапсулировать логику совершения комплексных операций над сущностями внутри своих методов, и таким образом расширить интерфейс Idao<T>. Выгода применения данного паттерна в том, что пользователю не нужно заботится о том, чтобы логировать операции напрямую, через класс-обертку DataManager.

* **Объектная модель EntityFramework**

В разработке слоя доступа к данным был использован подход Database-first, и в данном случае под объектной моделью EntityFramework понимается набор сгенерированных ядром EF классов.

* **Ядро EF**

В разработке использовался Entity Framework v 5.0.

### 3.4.2. Слой бизнес логики приложения

#### 3.4.2.1. Модуль компиляции

Модуль компиляции приложений представлен в виде класса Builder и набора классов-хелперов. На рисунке 3.5 представлена декомпозиция данного модуля.

SVN

Компонент компиляции

Веб- приложение

Svn Checkout

Рис. 3.5. Декомпозиция модуля компиляции

Создание скомпилированного пакета можно разделить на 3 этапа:

1. Выкачивание исходных файлов из репозитория
2. Компиляция
3. Сохранение пакета в директории Temp

Рассмотрим эти этапы и компоненты модуля компиляции, отвечающие за каждый из них подробнее.

* **Компонент SvnCheckout**

На этапе выкачивания исходных файлов, пользователь через веб-интерфейс оказывает управляющее воздействие на приложение, и в результате этого в ajax запросе ядру приложения передается информация о приложении, которое нужно скомпилировать. Взаимодействие с API SVN происходит через библиотеку SharpSvn (библиотека с открытым исходным кодом, разработанная компанией CollabNet). Скачанные исходные файлы сохраняются в файловой системе, а именно в том месте, которое задано в конфигурационном файле в секции AppSettings.

* **Компонент компиляции**

Модулю компиляции на вход подается путь, по которому расположены исходные файлы, путь к приложению MsBuild, который так же задан в конфигурационном файле, параметры компиляции приложения, а также, имя главного файла.

После успешной работы компилятора, скомпилированный пакет сохраняется в Temp директорию текущего пользователя, под которым был произведен вход в систему.

#### 3.4.2.2. Модуль развертывания приложений

Стадия развертывания может начаться только после завершения стадии компиляции. Логика данного модуля реализована в классе Deploy. Декомпозиция модуля развертывания представлена на рисунке 3.6.

Веб- приложение

Компонент развертывания

Rempte Agent

Rempte IIS

Удаленный сервер

Рис. 3.6. Декомпозиция модуля развертывания

Этапы стадии развертывания

1. Поиск скомпилированного пакета в директории Temp
2. Отправка пакета на удаленный сервер
3. Развертывание пакета службой Remote Agent на IIS

* **Компонент развертывания**

После завершения стадии компиляции ajax запросом в модуль развертывания передается информация о том приложении, которое нужно развернуть на удаленном сервере.

Модуль развертывания, на основе входных данных ищет в директории Temp текущего пользователя скомпилированный файл, и при отсутствии такового, генерирует InvalidOperationException с соответствующей информацией.

В случае успешного поиска, найденный скомпилированный пакет отправляется на удаленный сервер, где обрабатывается настроенной службой Remote Agent.

* **Remote Agent**

Remote Agent – это служба, которая установлена на удаленном сервере, и которая прослушивает определенный порт. RA принимает скомпилированный пакет, переданный из приложения, и разворачивает его на IIS удаленного сервера.

#### 3.4.2.3. Модуль формирования отчетов

В качестве технологии построения отчетов используется Sql Server Reporting Services [5]. Каждый отчет в системе состоит из нескольких составляющих: самого файла отчета, разработанного в Report Builder и хранимой процедуры, которая предоставляет данные для отчета. По требованию пользователя, отправляется запрос отчета на SSRS, который в свою очередь возвращает нужный файл отчета формата \*.rdl. Специальный контрол на aspx странице отображает этот файл и отсылает запрос на sql server, чтобы выполнить нужную хранимую процедуру для данного отчета. В итого, на странице отображается отчет, с которым можно совершать некоторые манипуляции, в частности, экспортировать его в документ, отправить этот отчет нескольким пользователям системы по email.

## 3.5. Алгоритм обновления приложений

Рассмотрим алгоритм обновления приложений с архитектурной точки зрения более подробно.

1. В конфигурационном файле приложения задается путь к папке на сервере приложений, куда будут сохраняться скачанные исходные файлы из репозитория контроля версий.

2. При скачивании исходников, происходит проверка на то, существует ли данная папка в файловой системе. Если существует, то происходит ее удаление, если нет – создается.

3. После скачивания исходников, запускается модуль компиляции, который сохраняет скомпилированный пакет в директории %Temp% с именем, которое складывается из имени сервера, на который будет происходить развертывание и имени самого объекта развертывания.

4. Модуль развертывания, берет скомпилированный пакет, и отправляет его на удаленный сервер, где специальная служба разворачивает его на локальном IIS.

## 3.6. Выбор технологий для разработки системы

Выбор технологий для разработки системы происходил по некоторым критериям, в частности:

* Удобство внедрения в текущую инфраструктуру предприятия
* Стоимость внедрения
* Сложность взаимодействия с другими программными комплексами предприятия

Даже самая качественная система не сможет эффективно работать, и ее поддержка будет достаточно сложной и дорогой, если при ее создании были выбраны инструменты, не соответствующие требованиям, предъявляемым к системе. Далее описаны технологии, применяемые при разработке системы.

### 3.6.1. Выбор платформы

Для разработки приложения была выбрана платформа .NET [6].

**Описание технологии**

Платформа .NET впервые была представлена компанией Microsoft в 2002 году и моментально стала одним из основных средств разработки современного программного обеспечения. Технология .NET включает в себя множество API – интерфейсов: ADO.NET, Entity Framework, LINQ, WPF, WCF, WF, ASP.NET, ASP.NET MVC и т.д.

До того, как Microsoft выпустила данную технологию, разработчики программного обеспечения в среде Windows применяли модель программирования COM. Технология COM (Component Object Model) позволяла разрабатывать библиотеки, которыми можно было пользоваться из других языков программирования. Например, модуль, написанный на C++ мог взаимодействовать с кодом, написанным на Visual Basic. Независимая природа COM была очень удобной, но недостатком была усложненная инфраструктура, хрупкая модель развертывания и возможность работать только под управлением операционной системы Windows. Несмотря на сложность COM модели, данная технология применялась в большинстве успешных программных продуктах, но сейчас, большинство продуктов, работающих под управлением Windows разрабатываются без применения COM. Вместо этого настольные приложения, веб – сайты, службы операционной системы строятся с помощью платформы .NET.

Целью создания .NET было обеспечения более мощной, гибкой и простой модели программирования, по сравнению с COM. .NET Framework – это платформа для разработки приложений для операционных систем семейства Windows и не только. К примеру, возможна разработка под такие ОС как Mac OS X и различные дистрибутивы Linux и Unix. Ниже приведен краткий перечень некоторых средств, поддерживаемых в .NET:

* + **Возможность взаимодействия с существующим кодом**

Эта возможность позволяет компоновать существующие двоичные компоненты COM и более новый код, написанный на платформе .NET.

* + **Поддержка многих языков программирования**

Приложения .NET можно разрабатывать с использованием любого количества языков программирования.

* + **Общий механизм исполнения программ**

Один из аспектов этого механизма – наличие хорошо определенного набора типов, которые хорошо понимает каждый язык, поддерживаемый в .NET.

* + **Языковая интеграция**

В .NET поддерживается межъязыковое наследование, межъязыковая обработка исключений и межъязыковая отладка программного кода.

* + **Упрощенная модель развертывания**

В отличие от COM, библиотеки .NET не регистрируются в системном реестре. Так же, модель .NET позволяет существовать на одном компьютере нескольким версиям одной dll.

**Компоненты .NET (CLR, CTS, CLS)**

С точки зрения программиста .NET представляет собой исполняющую среду и огромную библиотеку базовых классов. Исполняющая среда называется CLR (Common Language Runtime). Главная задача среды выполнения – автоматическая обработка объектов .NET. Так же CLR заботится об управлении памятью, координирует потоки приложения, выполняет проверку безопасности.

Другим важным компонентом платформы .NET является общая система типов CTS (Common Type System). В спецификации CTS полностью описаны типы данных и программные инструкции, которые поддерживаются на уровне CLR.

Отдельно взятый язык программирования, поддерживаемый в .NET может не поддерживать все средства, описанные в CTS. Поэтому была разработана общеязыковая спецификация CLS (Common Language Specification), в которой описано подмножество общих типов и инструкция, которые должны поддерживать все языки платформы .NET.

В дополнение к CLR в .NET существует библиотека базовых классов, которая доступна абсолютно всем языкам программирования платформы .NET. Эта библиотека, помимо инкапсуляции различных примитивов (потоки, системы визуализации и т.д.) так же обеспечивает поддержку большого количества служб. Отношение между CLR, CTS, CLS и библиотекой базовых классов представлено на рисунке 3.7.



Рис. 3.7. Отношение между CLR, CTS, CLS и библиотекой базовых классов

**Выполнение программного кода в среде CLR**

Платформа .NET предоставляет среду выполнения CLR, ключевая особенность которой (в отличии от других сред выполнения, к примеру, JVM) заключается в том, что CLR обеспечивает единый уровень выполнения, который поддерживается всеми языками платформы .NET. Основной механизм CLR физически реализован в виде сборки mscoree.dll. Когда происходи обращение к какой-либо сборке, библиотека mscoree.dll загружается автоматически и загружает нужную сборку в память. Основная задача общеязыковой среды выполнения – определение физического местоположения сборки и нахождение запрошенного типа в двоичном файле, затем CLR размещает найденный тип в памяти, преобразую связанный в ним IL – код в машинные команды и после этого выполняет код. Помимо загрузок сборок и создания специальных типов CLR взаимодействует с библиотекой базовых классов, которая хоть и разделена на отдельные сборки, но у них всех есть единая точка загрузки – mscorlib.dll. При построении приложений на платформе .NET доступ к этой сборке предоставляется автоматически. На рисунке 3.8 показано взаимодействие исходного кода, отдельного компилятора .NET и исполняющей средой .NET.

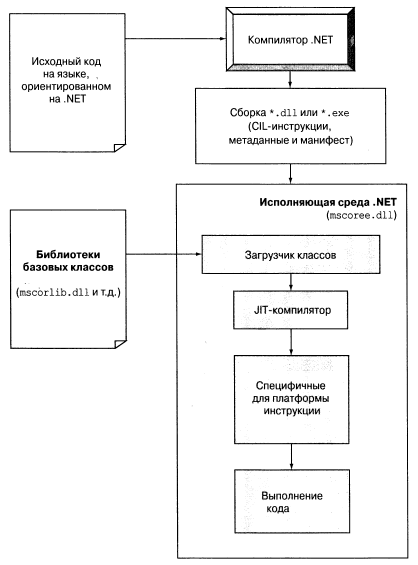


Рис. 3.8. Взаимодействие исходного кода, компилятора и исполняющей среды .NET

**Обоснование выбора технологии**

Выбор платформы .NET основывался прежде всего с тем, что большинство программных решений компании «ТНДМ Софт» разработано именно с использованием данной платформы, поэтому интеграция разработанного программного средства в устоявшуюся инфраструктуру произойдет наиболее безболезненно, если оно будет разработано на платформе .NET.

Платформа .NET предоставляет широкий стек технологий, которые удовлетворяют всем функциональным требованиям приложения, в частности, .NET имеет большие возможности именно в области веб – разработок, что является наиболее актуальным в разработке системы DOUIN.

### 3.6.2. Выбор языка программирования

В качестве языка программирования был выбран C#.

**Описание технологии**

C# является членом семейства C – подобных языков (куда так же входят Objective C, C++, Java и т.д.). К примеру, как и в Visual Basic C# поддерживает концепцию свойств, и, как и в C++ в C# можно перегружать операции, создавать перечисления, и функции обратного вызова. Так же, в C# встречаются конструкции языков функционального программирования, таких как List и Haskell, скажем, лямбда – выражения и анонимные типы. Ниже приведены некоторые характерные особенности языка C#, объединяющие все версии:

* + **Не требуется использование указателей**

В программах, написанных на C#, обычно не возникает потребности в использовании указателей, хотя в случае необходимости этот уровень можно опустить, использовав специальные конструкции языка.

* + **Автоматическое управление памятью**

В связи с этим, в языке нет потребности в ключевом слове delete.

* + **Перегрузка операций**

Как и C++, C# поддерживает возможность перегрузки операций для специальных типов.

* + **Поддержка программирования на основе атрибутов**

Это концепция разработки позволяет пометить объекты с помощью специального синтаксиса, и, таким образом, дополнительно уточнить поведение объекта в процессе выполнения программы.

Некоторые возможности более поздних версий C#:

* + **Возможность создания обобщенных типов**

Используя обобщения можно строить очень гибкий и безопасный к типам код, который определяет множество заполнителей.

* + **Возможность определения классов в нескольких файлах, за счет использования ключевого слова partial**
  + **Поддержка строго типизированных запросов (LINQ)**
  + **Включение лямба – операций**
  + **Новый синтаксис инициализации объектов**

Данная возможность позволяет устанавливать значения свойств во время создания объекта.

* + **Динамический поиск членов**

За счет использования ключевого слова dynamic, возможен вызов членов налету, независимо от инфраструктуры, с помощью которой они реализованы (COM, IronRuby и т.д.)

* + **Упрощенная модель параллельного программирования**

За счет ключевых слов async и await разработчикам стало проще писать многопоточные приложения.

**Обоснование выбора технологии**

Язык C# является основным языком платформы .NET, поэтому по этому языку больше всего справочной информации, чем по другим. Так же, все разработки под платформу .NET в компании «ТНДМ Софт» ведутся именно с использование данного языка. Поэтому, в целях простоты поддержки кода, для разработки приложения был выбран именно язык программирования C#.

### 3.6.3. Выбор технологии доступа к данным

В качестве технологии доступа к данным была выбрана технология Entity Framework. **Описание технологии** Entity Framework – ORM решение от Microsoft. То есть это объектно -ориентированная технология доступа к данным. Entity Framework предоставляет возможность взаимодействия с данными посредством LINQ to Entities или Entity SQL.

**Архитектура Entity Framework**

На рисунке 3.9 представлены основные компоненты архитектуры Entity Framework и их способы взаимодействия.

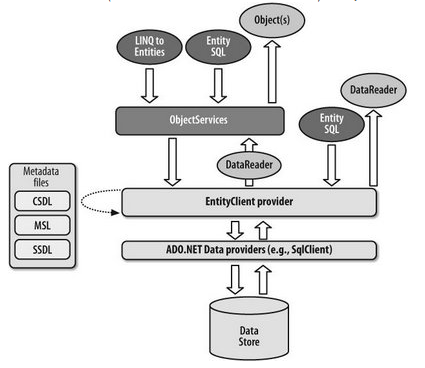


Рис. 3.9. Основные компоненты Entity Framework

В общем случае, выполнение запроса к базе данных выполняется следующим образом:

1. Формирование запроса к Entity Client посредством Object Services, либо напрямую
2. При помощи метаданных, Entity Client преобразует запрос LINQ to Entity, или Entity SQL в запрос SQL
3. SQL запрос отдается на выполнение ADO.NET провайдеру, который указан в строке подключения
4. Обработка возвращенных из СУБД данных.

Рассмотрим основные компоненты архитектуры Entity Framework более подробно:

**Entity Client**

Entity Client – это самый низкоуровневый способ создания запросов к EDM. Entity Client возвращает данные в виде строк и столбцов посредством EntityDataReader. Такая низкоуровневая реализация нужна для того, чтобы предоставить разработчикам возможность работы с такими конструкциями, которые не поддерживаются в Object Services, к примеру, некоторые типы хранимых процедур.

**Object Services**

Компонент Object Services располагается в пространстве имен System.Data.Objects и является верхушкой API Entity Framework. Object Services реализует удобный функционал для создания и взаимодействия сущностями концептуальной модели базы данных. Основной компонент Object Services – класс ObjectContext.

Основные задачи, которые решаются в Object Services:

* + **Обработка запросов**

Object Services позволяет генерировать запросы как при помощь LINQ to Entity, так и при помощи Entity SQL

* + **Материализация объектов**

После получения EntityDataReader от провайдера ADO.NET EntityClient возвращает его в ObjectServices, где происходит материализация данных в экземпляры сущностей

* + **Управление состоянием объектов**

ObjectContext для каждой сущности хранит ее состояние в виде ObjectStateEntry объекта, что позволяет отследить изменения сущности

* + **Управление взаимосвязями объектов**

Взаимосвязи между объектами обеспечивает ObjectContext

Entity Framework предоставляет для работы с данными несколько подходов: **CodeFirst –** подход использования Entity Framework, который позволяет сначала написать код, описывающий классы, а затем внутренний механизм в автоматическом режиме создаст базу данных. При чем отношения между таблицами будут распознаны автоматически. **Database First –** подход использования Entity Framework, который позволяет на основе существующей базы данных сгенерировать модели и контекст данных. Очень удобный подход, когда громоздкую базу данных необходимо представить в виде объектной модели. **Model First –** подход использования Entity Framework, который позволяет на основе модели, созданной в дизайнере EDM сгенерировать и базу данных и классы – модели.

**Обоснование выбора технологии**

С самой первой версии Entity Framework зарекомендовал себя как мощный ORM инструмент с возможностью очень гибкого манипулирования данными. За счет своей скорости и универсальности, Entity Framework способен сократить разработку компонентов приложения для доступа к данным в разы. Это и послужило поводом использовать данную технологию.

### 3.6.4. Выбор СУБД

В качестве системы управления базами данных для разработки приложения была выбрана СУБД MS SQL Server [7].

**Описание технологии**

MS Sql Server – это система управления реляционными базами данных, разработанная Microsoft. Основной язык написания запросов – Transact SQL, который разработан по стандартам ANSI/ISO SQL. Sql Server предоставляет широкие возможности по разработке и поддержке реляционных баз данных, а так же многие другие смежные технологии, например, в Sql Server реализован мощный подход к построению отчетов Sql Server Reporting Services.

**Обоснование выбора технологии**

Данная технология выбрана по причине удобной интеграцией с другими технологиями Microsoft, а также, потому, что Sql Server Reporting Services один из самых мощных движков построения отчетов на данный момент, существующих на рынке программного обеспечения.

### 3.6.5. Выбор технологии разработки веб-приложения

Для разработки самого веб приложения была выбрана технология ASP.NET MVC 4 [8].

**Описание технологии**

ASP.NET MVC – это платформа веб – разработок от Microsoft, которая сочетает в себе всю гибкость и аккуратность модели MVC. Благодаря архитектуре MVC, платформа ASP.NET MVC предоставляет значительное разделения ответственности.

Ключевые преимущества данной технологии:

1. **Адаптация к архитектуре MVC**

В ASP.NET MVC реализована современная модель паттерна MVC, которая отлично подходит для разработки современных веб – приложений.

1. **Расширяемость**

Внутренние компоненты ASP.NET MVC являются независимыми элементами целостной архитектуры. Это позволяет заменить практический любой внутренний механизм приложения, например, изменить логику поиска контроллеров, поиска маршрутов, или же изменить правила биндинга данных.

1. **Тестируемость**

За счет применения архитектуры MVC, разработчики могут проектировать приложения, которые легко тестировать с помощью. Разработчики ASP.NET MVC обеспечили для каждого компонента архитектуры замечательные возможности автоматического тестирования.

1. **Жесткий контроль над HTML**

В ASP.NET MVC учтена важность генерации ясного и легко поддерживаемого кода разметки. ASP.NET MVC позволяет разрабатывать легкие и элегантные пользовательские контролы за счет гибким контролем за HTML разметкой и стилями CSS. Так же, данная технология облегчает разработчикам использование открытых библиотек по проектированию пользовательского интерфейса, таких, например, как JQuery UI. Сгенерированные страницы ASP.Net не содержат объектов ViewState, поэтому эти страницы весят намного меньше, чем классические страницы WebForms.

1. **Мощная система маршрутизации**

Разработчики веб – приложений придают большое значение читабельным URL адресам. Это важно потому, что ключевые слова помогают логически разобрать URL. На ранних платформах чистые адреса было сложно использовать. ASP.Net MVC по умолчанию позволяет разработчикам пользоваться чистыми адресами. Это предоставляет полный контроль над схемой URL адресов.

Приложения ASP.NET MVC работают в связке с веб-сервером IIS. На рисунке 3.10 представлена общая схема обработки http запроса в IIS.



HTTP Protocol Stack (Http.sys)

Windows Activation Service (WAS)

World Wide Web Publishing Service (WWW Service)

W3WP.exe

Рабочий процесс

ApplicationHost.config

Svchost.exe

Application Pool

1

8

2

5

6

3

4

7

**INTERNET**

Рис 3.10. Обработка запроса в IIS

Рассмотрим более подробно обработку http запроса [9]:

1. **Браузер обращается к серверу по определенному адресу, и этот запрос перехватывает драйвер Http.sys**

Http.sys перехватывает все входящие запросы и переправляет их в нужный пул приложения. Если вдруг пул не доступен, то все запросы сохраняются в очереди, которая отводится для каждого пула. Таким образом, все входящие запросы пользователей никуда не деваются, даже если приложение временно не доступно. Так же Http.sys может кэшировать данные.

1. **Служба W3SVC**

Задачи данной службы:

* + Администрирование Http.sys;
  + Управление рабочими процессами;
  + Мониторинг показателей производительности веб – сайтов;

Данная служа функционирует в операционной системе в контексте процесса Svchost.exe. Начиная с IIS версии 7, функция управления процессами была вынесена в службу WAS.

1. **Служба WAS**

Основная задача службы WAS – управление рабочими процессами. Так же служба WAS считывает конфигурационный файл ApplicationHost.config, в котором задаются параметры приложения, к которому обратился пользователь по протоколу http. Данная служба запускает процесс W3WP.exe, в котором будет выполняться приложение, если этот процесс еще не запущен.

1. **Пул приложений**

Пул приложений призван изолировать приложения в отдельном участке памяти. Суть в том, что все приложения выполняются в специальном процессе W3WP.exe, и исключение в приложении приводит к краху только этого процесса.

1. **Результат запроса**

После всех шагов, результат возвращается драйверу Http.sys, который отдает этот результат браузеру пользователя.

Выше описана обработка http запроса с точки зрения веб – сервера IIS. На рисунке 3.11 представлен конвейер обработки запроса с точки зрения приложения ASP.NET MVC, которое выполняется в отдельном пуле приложений.

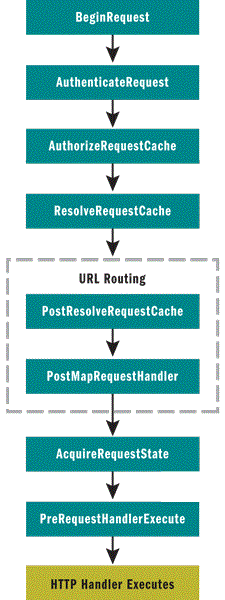


Рис 3.11. Конвейер обработки запроса ASP.NET MVC

Рассмотрим процесс обработки http запроса приложением ASP.NET MVC:

1. **Маршрутизация**

В после поступления сигнала в процесс приложения модуль UrlRoutingModule запускает систему маршрутизации. Задача системы маршрутизации – проанализировать входящую строку запроса. Систему маршрутизации проверяет в первую очередь соответствует ли входящий запрос какому-либо файлу, который уже существует на диске. Если да, то маршрутизация завершает свою работу.

1. **Таблица маршрутизации**

Таблица маршрутизации хранится в специальной коллекции. Для поиска соответствия входящего URL модуль маршрутизации сканирует таблицу маршрутизации начиная с самого первого элемента. В случае успешного поиска, система маршрутизации передает управлению в обработчик маршрута, так же в обработчик передается контекст запроса.

1. **Поиск контроллера**

MvcRouteHandler знает, как с помощью контекста запроса вызвать нужный контроллер. В работу вступает фабрика контроллеров. Фабрика контроллеров – это специальный механизм, задачей которого является инстанцирование нужного файла контроллера. По умолчанию в ASP.NET MVC используется фабрика контроллеров, которая называется DefaultControllerFactory.

1. **Поиск нужного действия контроллера**

После инстанцирования контроллера, в нем будет осуществляться поиск метода – действия, который должен вернуть результат в виде объекта ActionResult, в котором будет вызван метод ExcecuteResult, который непосредственно занимается формированием разметки.

1. **Ответ браузеру**

После формирования результата запроса в объекте ActionResult, управление передается драйверу Http.sys, который отдает результат http запроса браузеру пользователя.

**Обоснование выбора технологии**

Технология ASP.NET MVC была выбрана по причине возможности гибкого конфигурирования процесса распознания входящих URL адресов, что очень важно для решения поставленной задачи. Так же, ASP.NET MVC упрощает разработку пользовательского интерфейса в связке с JQuery UI, элементы которого активно используются в разработанном приложении.

### 3.6.6. Выбор технологий построения пользовательского интерфейса

В качестве технологий построения пользовательского интерфейса были выбраны технологии jQuery [10] и Twitter Bootstrap [11].

#### 3.6.6.1. jQuery

**Описание технологии** jQuery – это javascript фрэймвок, который призван упростить использование классического javascript [12], что позволяет повысить эффективность труда разработчиков за счет многоразового использования кода, а также, программистам нет необходимости беспокоиться о вопросах кроссбраузерной разработки, так как все эти механизмы реализованы в самом ядре jQuery. Ниже описаны основные функции jQuery:

* Обращение и манипулирование любым элементом DOM;
* Возможность работы с событиями;
* Упрощенная модель работы с ajax;
* Наличие огромного количества javascript – плагинов, предназначенных для создания элементов пользовательского интерфейса.

Точно так же, как CSS отделяет визуализацию от структуры HTML, JQuery отделяет поведение от структуры HTML. Например, вместо прямого указания на обработчик события нажатия кнопки управление передаётся JQuery, которая идентифицирует кнопки и затем преобразовывает его в обработчик события клика. Такое разделение поведения и структуры также называется принципом ненавязчивого JavaScript.

Библиотека jQuery содержит функциональность, полезную для максимально широкого круга задач. Тем не менее, разработчиками библиотеки не ставилась задача совмещения в jQuery функций, которые подошли бы всюду, поскольку это привело бы к большому коду, большая часть которого не востребована. Поэтому была реализована архитектура компактного универсального ядра библиотеки и плагинов. Это позволяет собрать для ресурса именно ту JavaScript-функциональность, которая на нём была бы востребована.

**Обоснование выбора**

Данная технология была выбрана по причине удобства использования всей мощи классического javascript через упрощенную модель программирования jQuery. Использование jQuery в веб – разработках является признанным стандартом, и использование данной библиотеки способствует сокращению времени разработки.

#### 3.6.6.2. Twitter Bootstrap

**Описание технологии** Twitter Bootstrap – свободный CSS фреймворк от компании «Twitter Inc». После выхода самой первой версии, вместе с CSS так же стали распространятся javascript расширения. Bootstrap использует самые современные наработки в области CSS и HTML.

Основные инструменты Bootstrap:

* **Сетки** — заранее заданные размеры колонок, которые можно сразу же использовать, например, ширина колонки 140px относится к классу .span2, который можно использовать в CSS описании документа;
* **Шаблоны** — Фиксированный или резиновый шаблон документа;
* **Типографика** — Описания шрифтов, определение некоторых классов для шрифтов таких как код, цитаты и т. п.;
* **Медиа** — Представляет некоторое управление изображениями и Видео;
* **Таблицы** — Средства оформления таблиц, вплоть до добавления функциональности сортировки;
* **Формы** — Классы для оформления не только форм, но и некоторых событий, происходящих с ними;
* **Навигация** — Классы оформления для Табов, Вкладок, Страничности, Меню и Тулбара;
* **Алерты** — Оформление диалоговых окон, Подсказок и Всплывающих окон;

**Обоснование выбора** Главная причина выбора данной технологии – упрощение применения современных достижений в области веб – дизайна при разработке пользовательского интерфейса, что позволяет сократить время разработки.

### 3.6.7. Выбор технологии построения отчетов

В качестве системы построения отчетов была выбрана технология Sql Server Reporting Services [5]

**Описание технологии SQL Server Reporting Services** — программная серверная система создания отчётов, разработанная корпорацией Microsoft. Она может быть использована для подготовки множества интерактивных и печатных отчётов. Система администрируется через веб-интерфейс. Reporting Services используют интерфейс веб-служб для поддержки разработки обычных отчётных приложений. SSRS соперничает с Crystal Reports и другими BI-инструментами, и входит в состав Express, Workgroup, Standard и Enterprise версий Microsoft SQL Server в качестве устанавливаемого дополнения. Reporting Services были впервые выпущены в 2004 году как дополнение для SQL Server 2000. Вторая версия была выпущена в виде составной части SQL Server 2005 в ноябре 2005 года. Следующая версия была выпущена как часть SQL Server 2008 в августе 2008. Последняя на данный момент версия была выпущена как часть SQL Server 2012 в марте 2012.

В SSRS отчёты описываются при помощи Report Definition Language (RDL) на языке разметки XML. отчёты могут проектироваться при помощи последних версий Microsoft Visual Studio (включая Visual Studio.NET 2003 и Visual Studio 2005) с входящим в них дополнением Business Intelligence Projects или при помощи входящего в комплект Report Builder — упрощённого инструмента, не предлагающего полного функционала Visual Studio. отчёты, определенные при помощи RDL, могут создаваться во множестве различных форматов, включая Excel, PDF,CSV, XML, TIFF (и других графических форматах), а также HTML Web Archive. SQL Server 2008 SSRS также может подготавливать отчёты в формате Microsoft Word (DOC). Сторонние генераторы отчётов предлагают возможность создания отчётов в дополнительных форматах. Пользователи могут работать с веб-службой Report Server напрямую или использовать Report Manager — веб-приложение, взаимодействующее с веб-службой Report Server. При помощи Report Manager могут просматривать и управлять отчётами, также, как и управлять и оперировать источниками данных и настройками безопасности. Отчёты могут рассылаться по электронной почте или записываться на файловую систему как обычный файл. Защита выполняется на основе ролей и может накладываться на отдельные элементы, как например, отчёт или источник данных, каталог элементов или сайт вообще. Роли безопасности и права являются наследуемыми и могут быть переопределены. В дополнение к использованию отдельного Report Server, поставляемого с SQL Server, RDL-отчёты можно просматривать при помощи веб-контрола ASP.NET ReportViewer или Windows Forms-контрола ReportViewer. Это позволяет встраивать отчёты прямо в веб-страницы или .NET-приложения. Контрол ReportViewer обрабатывает отчёты одним из двух способов: (a) на стороне сервера, где отчёт обрабатывается Report Server; и (b) локальная обработка, где соответствующий контрол самостоятельно обрабатывает RDL-файл. SQL 2005 reporting services также поддерживает обработку произвольных отчётов: разработчик создает схему отчёта и развертывает её на сервере отчётности, где пользователь может выбирать нужные поля/данные и создавать отчёты в соответствии со своими нуждами. Потом пользователи могут загружать отчёты локально.

**Обоснование выбора технологии**

Основная причина использование технологии SSRS в качестве инструмента построения отчетов – удобство интеграции с .NET приложениями, в частности, с ASP.NET приложениями за счет использования контрола ReportViewer.

## 3.7. Описание классов системы

Классы – основное компонент любой объектно - ориентированной архитектуры. По диаграмме классов всегда можно понять структуру приложения. С целью облегчения восприятия архитектуры системы в целом, было принято отойти от классического способа представления диаграммы классов, отобразив только связи между слоями данных. Диаграмма представлена на рисунке 3.12.

BuildParametersRepository

CustomizationRepository

DeployObjectsTypesRepository

DeployParametersRepository

NotificationsApplicationsRepository

NotificationsRepository

OperationsRepository

ReportsRepository

ServerObjectsRepository

ServersRepository

ServerTypesRepository

SvnParametersRepository

VersionsHistoryRepository

Слой доступа к данным

CsProjSetting

NotificationModel

ReportModel

ReportParameter

ReportSettingsModel

ServerModel

ServerObjectModel

Модели данных

HomeController

NotificationsController

ReportsController

ReportsSettingsController

ServerController

ServerObjectsController

SettingsController

Контроллеры

Рис 3.12. Диаграмма классов системы

Рассмотрим составляющие всех слоев более подробно.

**Слой доступа к данным:**

Классы этого слоя представляют собой конкретную реализацию логики доступа к данным в базе данных.

*BuildParametersRepository* – класс, описывающий работу с параметрами компиляции, которые хранятся в базе данных.

*CustomizationRepository* – класс, описывающий работу с элементами кастомизации, которые хранятся в базе данных.

*DeployObjectTypesRepository* – класс, описывающий работу с типами объектов развертывания, которые хранятся в базе данных.

*DeployParametersRepository –* класс, описывающий работу с параметрами развертывания, которые хранятся в базе данных.

*NotificationsRepository –* класс, описывающий работу с объектами оповещения, которые хранятся в базе данных.

*NotificationsApplicationsRepository –* класс, описывающий работу со связями объектов сервера и объектов оповещения.

*OperationsRepository –* класс, описывающий работу с объектами операций в системе.

*ReportsRepository –* класс, описывающий работу с отчетами, которые хранятся в базе данных.

*ServerObjectsRepository –* класс, описывающий работу с объектами серверов.

*ServersRepository –* класс, описывающий работу с объектами серверов.

*ServerTypesRepository –* класс, описывающий работу с типами серверов, которые хранятся в базе данных.

*SvnParametersRepository –* класс, описывающий работу с параметрами SVN, которые хранятся в базе данных.

*VersionsHistoryRepository –* класс, описывающий работу с историей изменения версии приложения на продуктивном сервере.

**Модели данных:**

Модели данных предоставляют собой конкретные сущности, с которыми взаимодействует пользователь.

*CsProjSettings –* модель параметров файла проекта.

*NotificationModel –* модель объекта оповещения.

*ReportModel –* модель отчета.

*ReportParameter –* модель параметра отчета.

*ServerModel –* модель сервера.

*ServerObjectModel –* модель объекта сервера.

**Контроллеры:**

Каждый контроллер представляет собой логику обработки данных одной из моделей, либо реализует общий функционал доступа к системе.

*HomeController –* стартовый контроллер, представляет главную страницу.

*NotificationsController –* контроллер обработки моделей оповещения.

*ReportsController –* контроллер обработки моделей отчетов.

*ReportsSettingsController –* контроллер обработки моделей настройки отчетов.

*ServerController –* контроллер обработки моделей серверов.

*ServerObjectsController –* контроллер обработки моделей объектов серверов.

*SettingsController –* контроллер обработки моделей настроек приложения.

## 3.8. Проектирование базы данных системы

Одним из наиболее важных и сложных этапов разработки программного обеспечения является этап проектирования базы данных. На данном этапе необходимо решить вопрос о наиболее эффективной структуре данных. Основные цели, которые при этом преследуются:

* обеспечение быстрого доступа к данным;
* отсутствие избыточности данных;
* целостность данных.

В разработанной базе данных используются следующие сходные данные:

* информация о серверах;
* информация об объектах серверов;
* историческая информация об изменении ПО на серверах;
* параметры развертывания приложений;
* параметры компиляции приложений;
* информация об отчетах, добавленных в систему;
* информация о типах объектов серверов;
* информация о действиях пользователей в системе;
* информация об объектах оповещений.

Проведя анализ предметной области и требований к работе системы, была разработана структура базы данных, приведенная на диаграмме базы данных. Диаграмма показана на рис 3.13.

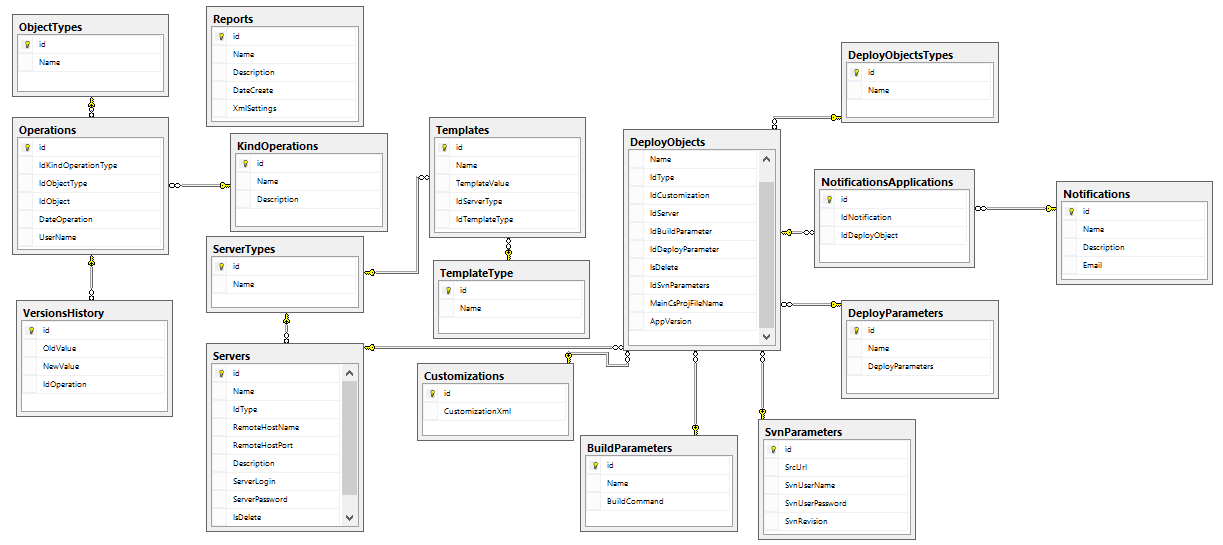


Рис 3.13. Диаграмма базы данных

Рассмотрим назначение и структуру каждой таблицы в отдельности:

1. Таблица «Notifications».

В данной таблице хранится информация об объектах оповещения, то есть имена и адреса электронной почты.

Таблица 3.1. Описание таблицы «Notifications»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Имя |
| Description | Текстовый | Описание объекта |
| Email | Текстовый | Адрес электронной почты |

2. Таблица «DeployObjects».

В данной таблице хранится информация обо всех объектах серверов, зарегистрированных в системе.

Таблица 3.2. Описание таблицы «DeployObjects»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название объекта |
| IdType | Целочисленный | Идентификатор типа объекта |
| IdCustomization | Целочисленный | Идентификатор кастомизации объекта |
| IdServer | Целочисленный | Идентификатор сервера |
| IdBuildParameter | Целочисленный | Идентификатор параметров компиляции |
| IdDeployParameter | Целочисленный | Идентификатор параметров развертывания |
| IsDelete | Целочисленный | Признак удаления |
| IdSvnParameter | Целочисленный | Идентификатор параметров системы контроля версий |
| MainCsProjFileName | Текстовый | Имя главного файла для компиляции |
| AppVersion | Текстовый | Текущая версия приложения |

3. Таблица «NotificationsApplications».

Данная таблица является таблицей – связью между таблицами «DeployObjects» и «Notifications». Реализует связь многие ко многим.

Таблица 3.3. Описание таблицы «NotificationsApplications»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| IdNotification | Целочисленный | Идентификатор объекта оповещения |
| IdDeployObject | Целочисленный | Идентификатор объекта сервера |

4. Таблица «DeployParameters».

В данной таблице хранятся параметры для компиляции объекта сервера.

Таблица 3.4. Описание таблицы «DeployParameters»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название параметра |
| DeployParameters | Текстовый | Значение параметра |

5. Таблица «DeployObjectsTypes».

Данная таблица содержит в себе список типов объектов серверов.

Таблица 3.5. Описание таблицы «DeployObjectsTypes»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название типа |

6. Таблица «SvnParameters».

Данная таблица содержит информацию о репозитории системы контроля версии для конкретного объекта сервера.

Таблица 3.6. Описание таблицы «SvnParameters»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| SrcUrl | Текстовый | Адрес репозитория |
| SvnUserName | Текстовый | Имя пользователя SVN |
| SvnUserPassword | Текстовый | Пароль пользователя VN |
| SvnRevesion | Целочисленный | Ревизия, с которой нужно выкачать исходные файлы |

7. Таблица «BuildParameters».

Данная таблица содержит информацию о параметрах компиляции приложения.

Таблица 3.7. Описание таблицы «BuildParameters»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название параметра |
| BuildCommand | Текстовый | Значение параметра |

8. Таблица «Customizations».

Данная таблица содержит в себе информацию о кастомизации объекта сервера.

Таблица 3.8. Описание таблицы «Customizations»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название типа |

9. Таблица «Servers».

Данная таблица содержит в себе информацию обо всех продуктивных серверах, зарегистрированных в системе.

Таблица 3.9. Описание таблицы «Servers»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название объекта |
| IdType | Целочисленный | Идентификатор типа объекта |
| RemoteHostName | Текстовый | Адрес удаленного хоста |
| RemoteHostPort | Целочисленный | Порт удаленного хоста |
| Description | Текстовый | Описание |
| ServerLogin | Текстовый | Имя учетной записи, под которой происходит подключение |
| ServerPassword | Целочисленный | Пароль учетной записи, под которой происходит подключение |
| IsDelete | Целочисленный | Признак удаления |

10. Таблица «ServerTypes».

Данная таблица содержит в себе список типов серверов.

Таблица 3.10. Описание таблицы «ServerTypes»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название типа |

11. Таблица «Templates».

Данная таблица содержит в себе базовые настройки для конкретного типа сервера.

Таблица 3.11. Описание таблицы «Templates»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название объекта |
| IdType | Целочисленный | Идентификатор типа объекта |
| RemoteHostName | Текстовый | Адрес удаленного хоста |
| RemoteHostPort | Целочисленный | Порт удаленного хоста |

12. Таблица «TemplateType».

Данная таблица содержит в себе список типов шаблонов для каждого конкретного типа сервера.

Таблица 3.12. Описание таблицы «TemplateType»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название типа |

13. Таблица «Reports».

Данная таблица содержит информацию обо всех отчетах, добавленных в систему.

Таблица 3.13. Описание таблицы «Reports»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название отчета |
| Description | Текстовый | Описание |
| DateCreate | Дата | Дата добавления отчета в систему |
| XmlSettings | XML | Настройка отчета |

14. Таблица «Operations».

Данная таблица содержит информацию обо всех действиях пользователей в системе над серверами и их приложениями.

Таблица 3.14. Описание таблицы «Operations»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| IdKindOperationType | Целочисленный | Идентификатор типа операции |
| IdObjectType | Целочисленный | Идентификатор типа объекта |
| IdObject | Целочисленный | Идентификатор объекта |
| DateOperation | Дата | Дата совершения операции |
| UserName | Текстовый | Пользователь, совершивший операцию |

15. Таблица «KindOperations».

Данная таблица содержит в себе список возможных операций над объектами системы

Таблица 3.15. Описание таблицы «KindOperations»

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование поля | Тип данных |  | Описание |
| Id | Целочисленный |  | Идентификатор |
| IdKindOperationType | Целочисленный |  | Идентификатор типа операции |
| IdObjectType | Целочисленный |  | Идентификатор типа объекта |

16. Таблица «VersionsHistory».

Данная таблица содержит в себе историю изменения версий приложений.

Таблица 3.16. Описание таблицы «VersionsHistory»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| IdKindOperationType | Целочисленный | Идентификатор типа операции |
| IdObjectType | Целочисленный | Идентификатор типа объекта |
| IdObject | Целочисленный | Идентификатор объекта |
| DateOperation | Дата | Дата совершения операции |

17. Таблица «ObjectTypes».

Данная таблица содержит в себе список типов объектов, над которыми совершались операции.

Таблица 3.17. Описание таблицы «ObjectTypes»

|  |  |  |
| --- | --- | --- |
| Наименование поля | Тип данных | Описание |
| Id | Целочисленный | Идентификатор |
| Name | Текстовый | Название типа |

## 3.9. Описание графического интерфейса системы

Доступ к функционалу приложения предоставляется через веб – интерфейс. Набрав в адресной строке браузера адрес нужного сервера, пользователь попадает на главную страницу приложения. Если пользователь еще не авторизован, ему будет предложено ввести пароль и имя пользователя. После успешного прохождения стадии авторизации, пользователю будет показано страница, которая представлена на рисунке 3.14.

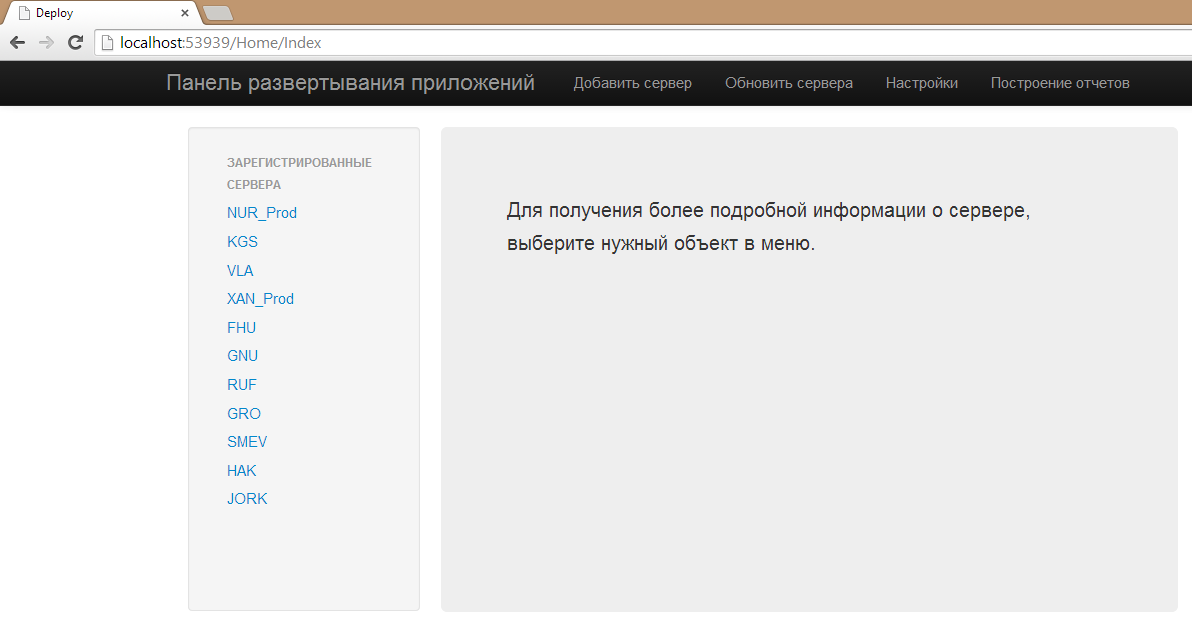


Рис. 3.14. Стартовая страница приложения

Слева отображается список уже добавленных в систему серверов, и чтобы посмотреть более подробную информацию об этом сервере, необходимо кликнуть левой клавишей мыши по его названию. После этого пользователь сможет увидеть больше информации об этом объекте (Рисунок 3.15).

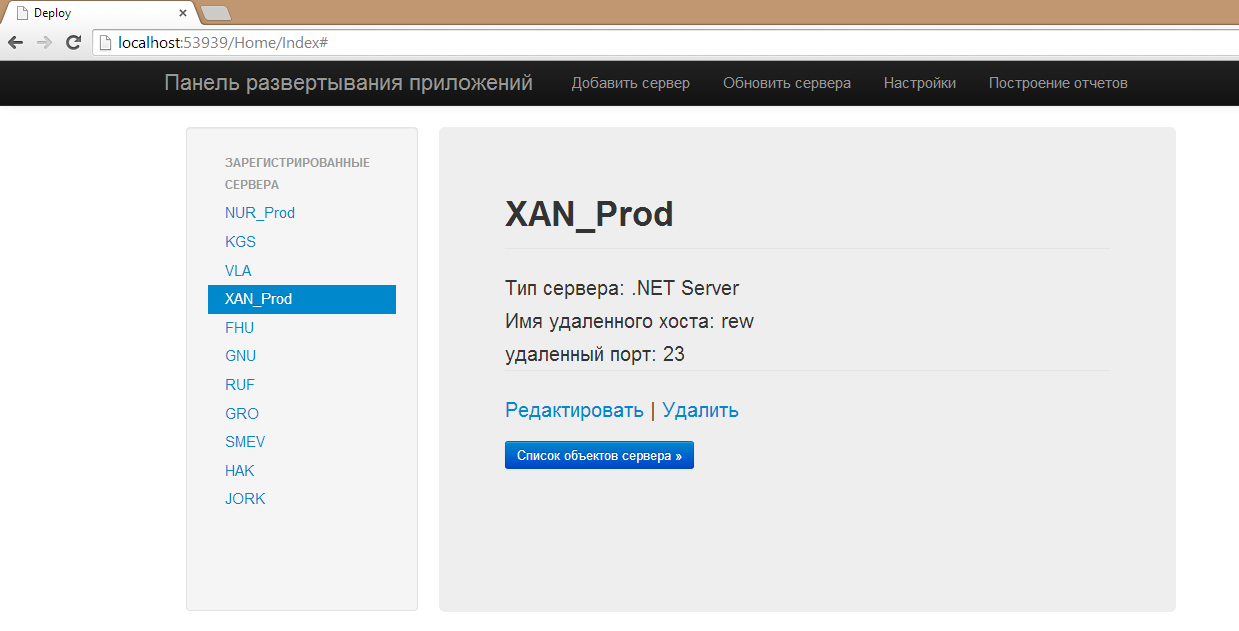


Рис 3.15. Информация о сервере

### 3.9.1 Добавление нового сервера

Для добавления нового сервера в систему, необходимо в верхнем меню перейти по ссылке «Добавит сервер». Пользователю откроется окно добавления нового сервер (Рис 3.16).

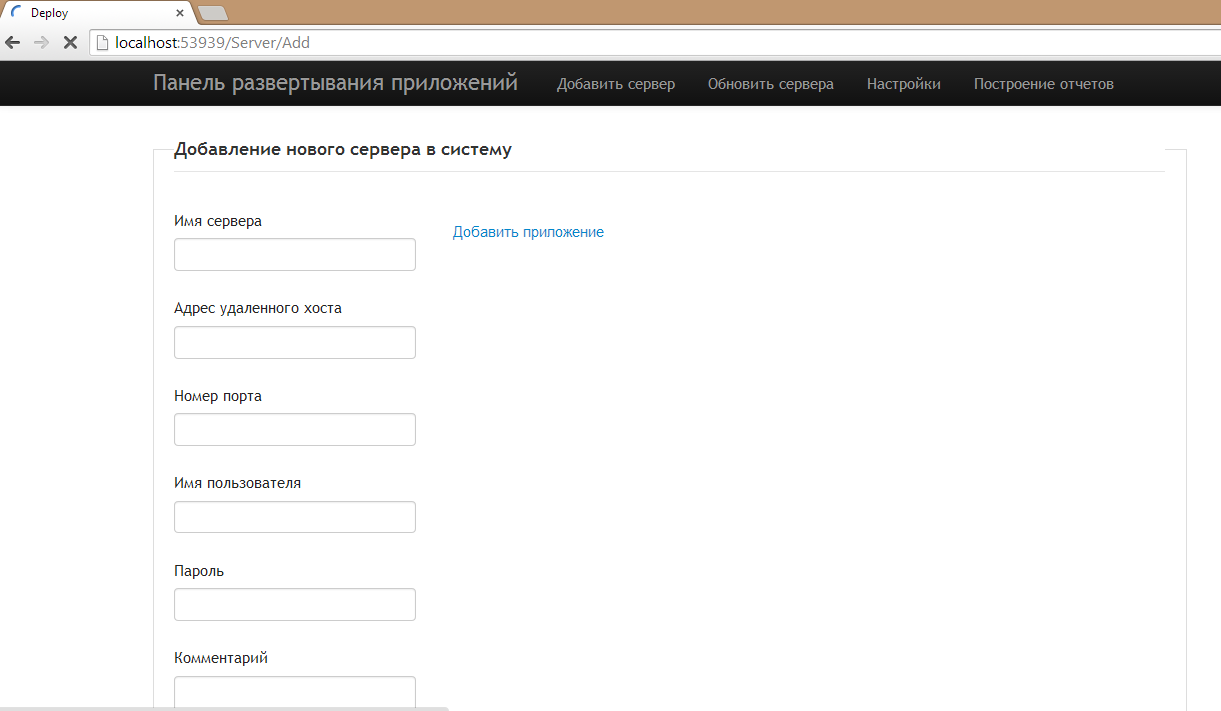


Рис 3.16. Страница добавления нового сервера

После заполнения всех полей сервера, к нему нужно добавить приложения, которые будут обновляться. Для этого нужно кликнуть по ссылке «Добавить приложение» и откроется модельное окно добавления приложения (Рис 3.17).

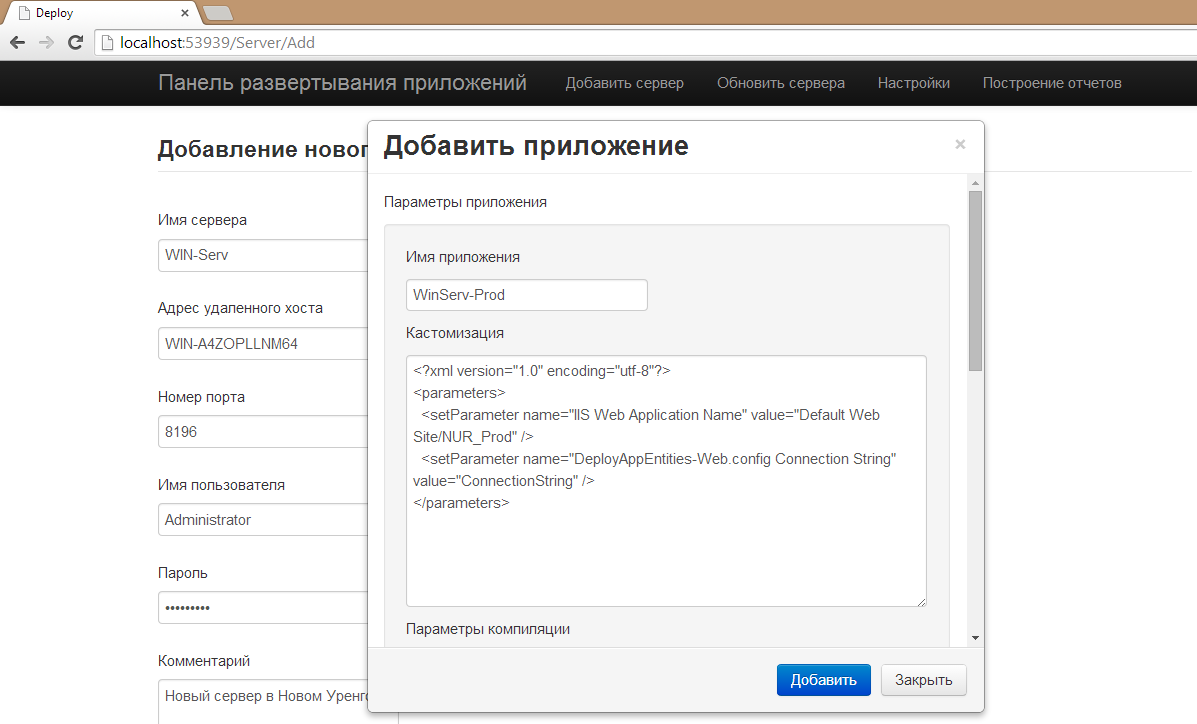


Рис 3.17. Модальное окно добавления нового приложения

После ввода всех параметров приложения, необходимо нажать на кнопку «Добавить». Таким образом можно добавить несколько приложений к одному серверу. В нашем случае их будет два – тестовое и продуктивное.

После добавления обоих приложений окно создания нового сервера будет иметь вид как на рисунке 3.18.

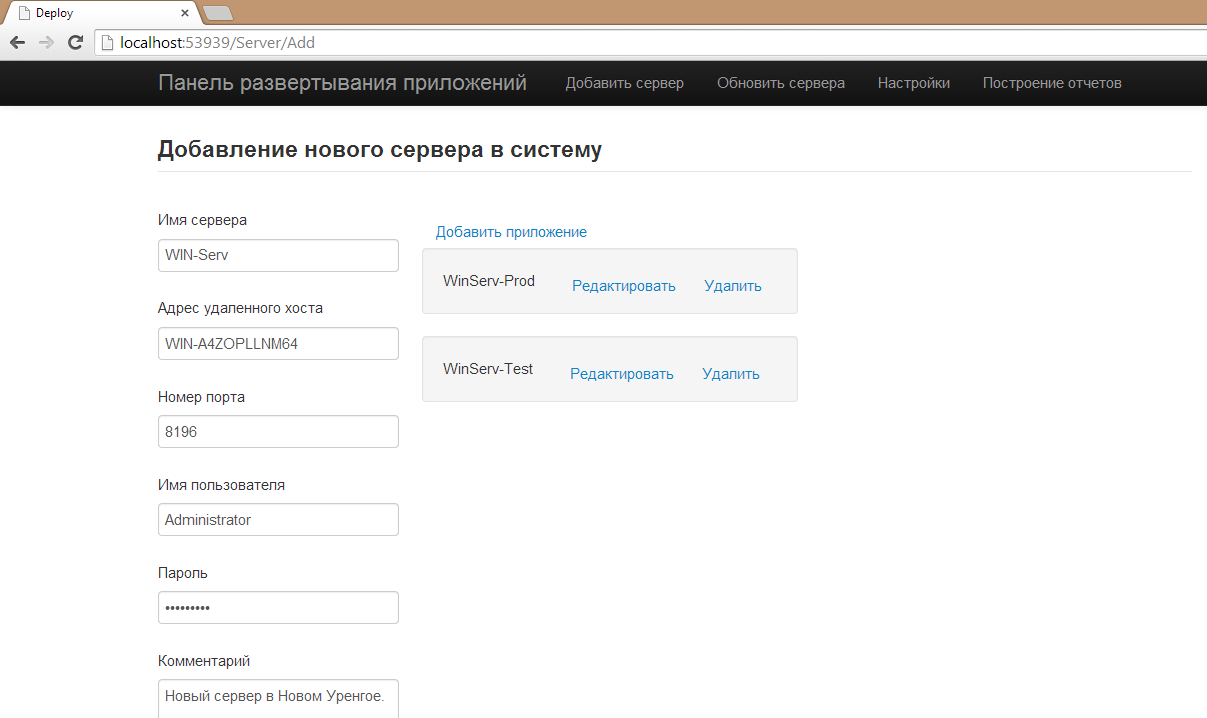


Рис 3.18. Окно добавления сервера с заполненными параметрами

После нажатия на кнопку «Добавить», новый сервер появится в списке серверов на стартовой странице. Для просмотра всех приложений этого сервера необходимо сначала нажать на его название в списке, а затем, в открывшейся информации о сервере нажать на кнопку «Список объектов сервера». После этого откроется модальное окно со списком приложений данного сервера (Рис 3.19).

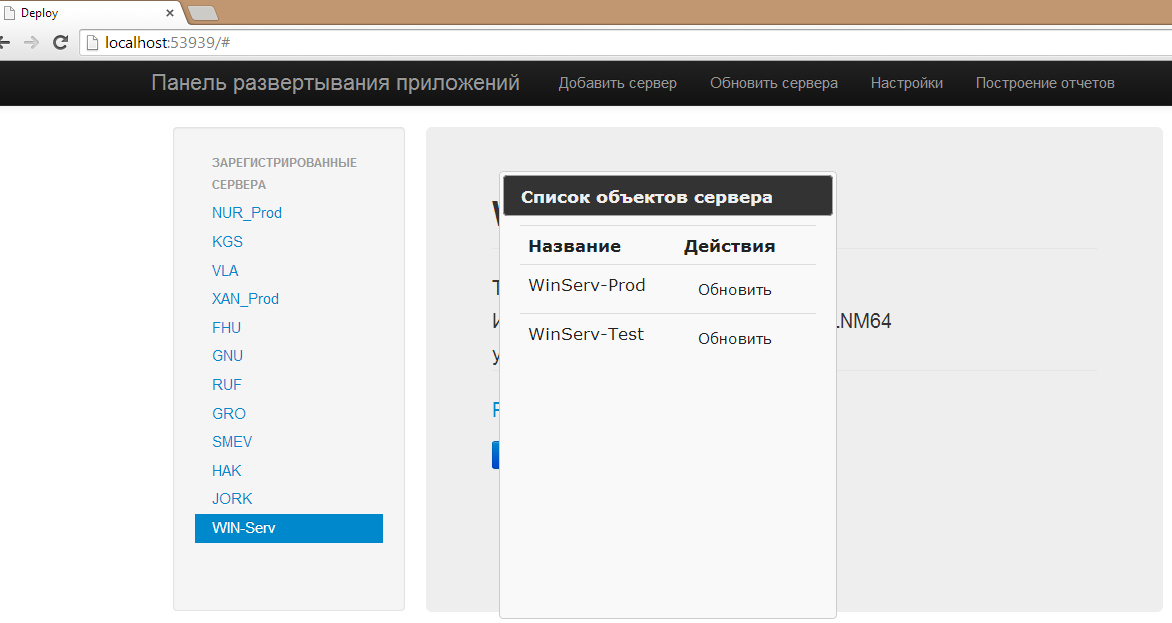


Рис 3.19. Список объектов сервера

Для перехода к конкретному объекту сервера, необходимо два раза кликнуть по его названию в диалоговом окне. После этого откроется новое окно с подробной информацией о текущем объекте (Рис. 3.20).

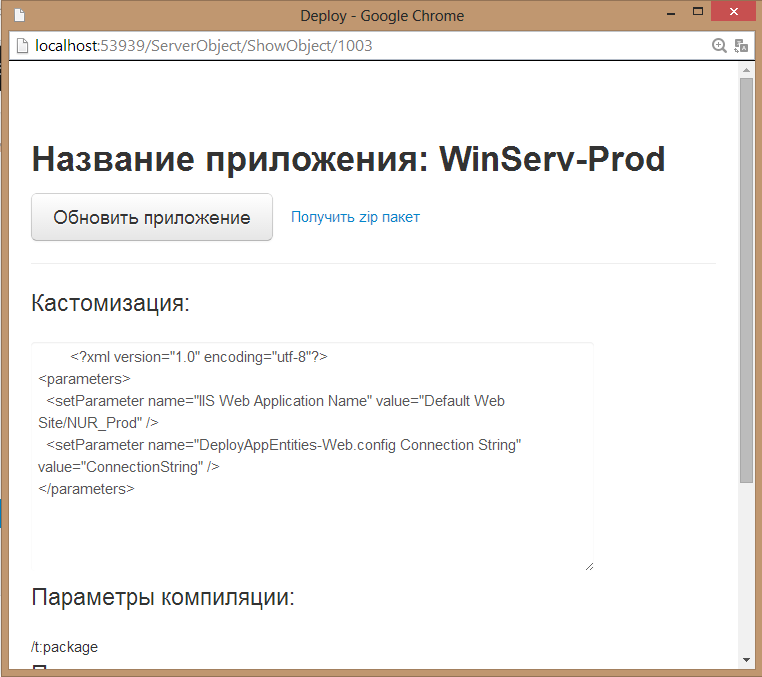


Рис 3.20. Окно объекта сервера

### 3.9.2. Конфигурирование объектов оповещения

Функционал приложения позволяет информировать пользователей системы об обновлениях приложения на сервере. Чтобы это настроить, необходимо сначала добавить в систему пользователей, которых необходимо оповещать. Для этого в верхней части главного окна нужно нажать на ссылку «Настройки» и в открывшемся окне выбрать пункт «Объекты оповещения». Если в системе до этого были добавлены некоторые объекты, они отобразятся на странице в виде таблицы. Для добавления нового, необходимо перейти по ссылке «Добавить» и откроется окно добавления нового объекта оповещения, у котором нужно заполнить все необходимые поля (Рис 3.21).

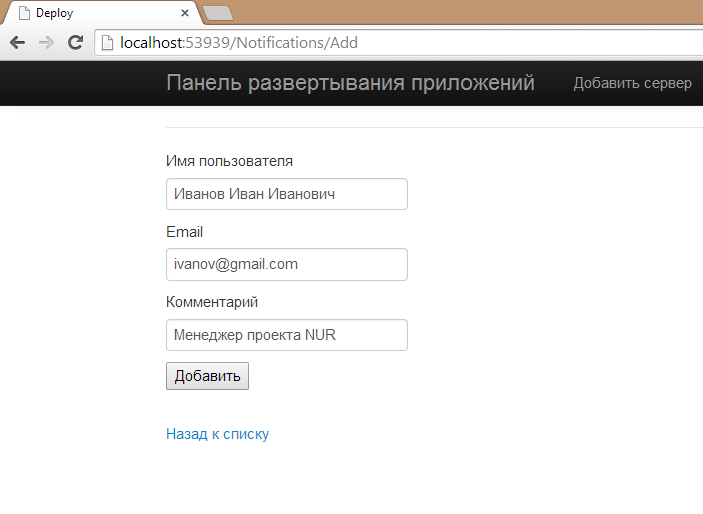


Рис 3.21. Добавление нового объекта оповещения

Описанные выше действия необходимо повторить для всех объектов, которые планируется добавить в систему. Добавленные объекты можно просмотреть на странице списка объектов оповещения (Рис 3.22).

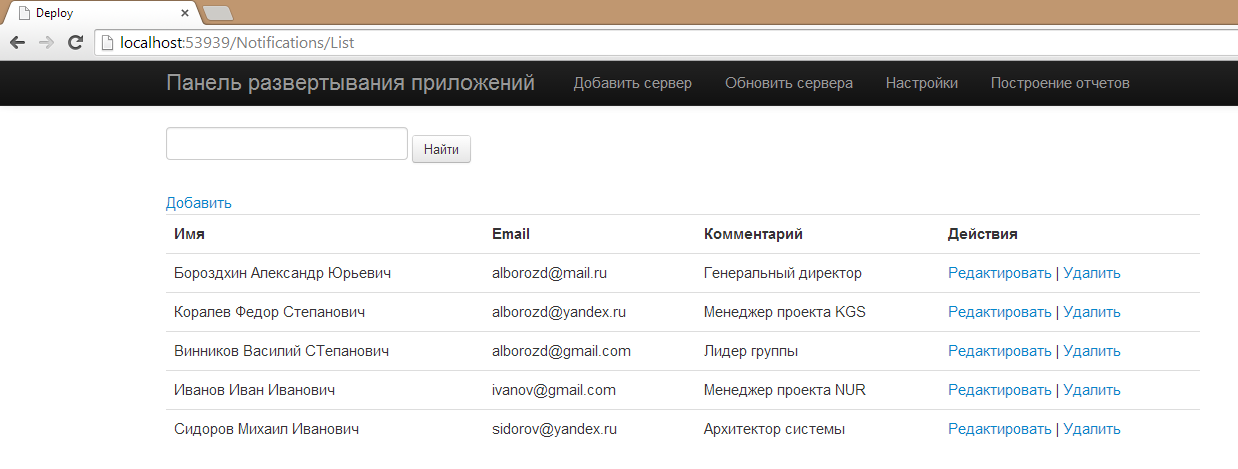


Рис 3.22. Список объектов оповещения

Теперь добавленные объекты оповещения необходимо добавить в ассоциацию с конкретным приложением. Для этого нужно перейти в режим редактирования объекта сервера (в нижней части окна информации об объекте сервера нажать на ссылку «Редактировать») и в самом низу открывшегося окна переместить нужные объекты из левой области ввода в правую и нажать на кнопку сохранить (Рис 3.23).

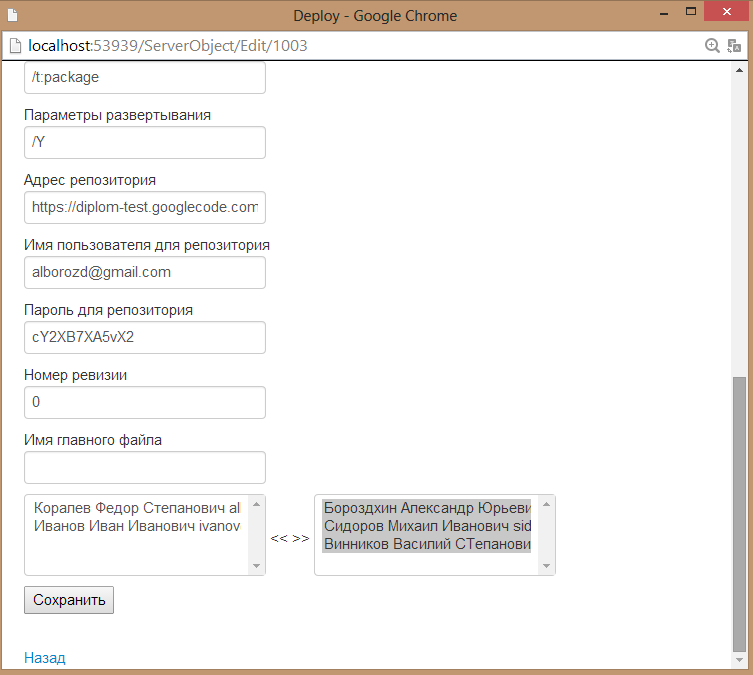


Рис 3.23. Ассоциирование объектов оповещения с конкретным приложением

Теперь после каждого обновления данного приложения, на электронную почту добавленных выше лиц будет приходить письмо об этом событии.

### 3.9.3. Развертывание приложения

Для удаленного развертывания необходимо в окне просмотра информации об объекте сервера нажать на кнопку «Обновить приложение». Перед запуском процесса обновления, необходимо ввести номер версии приложения, которое подвергается обновлению. После этого в новом диалоговом окне запустится процесс обновления (Рис 3.24).

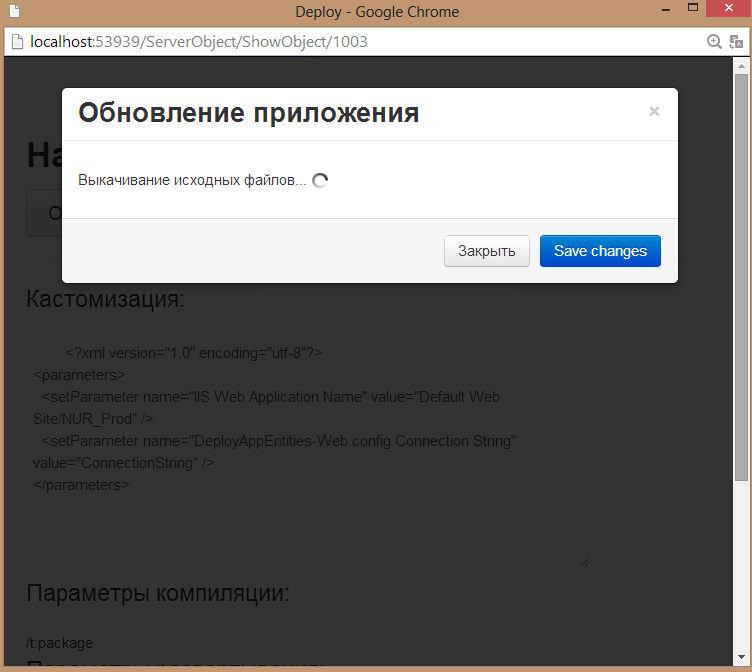


Рис 3.24. Начало обновления

### 3.9.4. Работа с отчетами

Для получения статистической информации можно воспользоваться разделом «Построение отчетов» после перехода на который, пользователю представится список отчетов, добавленных в систему.

#### 3.9.4.1. Разработка отчетов

Разработка любого отчета ведется в несколько этапов:

* Разработка хранимой процедуры (или любого другого источника данных для отчета);
* Разработка шаблона отчета.

Разработка шаблонов отчета ведется в специальном инструменте ReportBuilder (рис. 3.25), который поставляется вместе с Sql Server Reporting Services всех версий, кроме Express. Любой шаблон представляет из себя набор элементов, которые группируют данные, поставляемые источником данных (хранимая процедура, sql запрос и т.д.). В системе DOUIN изначально реализовано несколько отчетов, которые будут поставляться с приложением, но также возможно добавление новых отчетов в систему (данный функционал описан ниже).

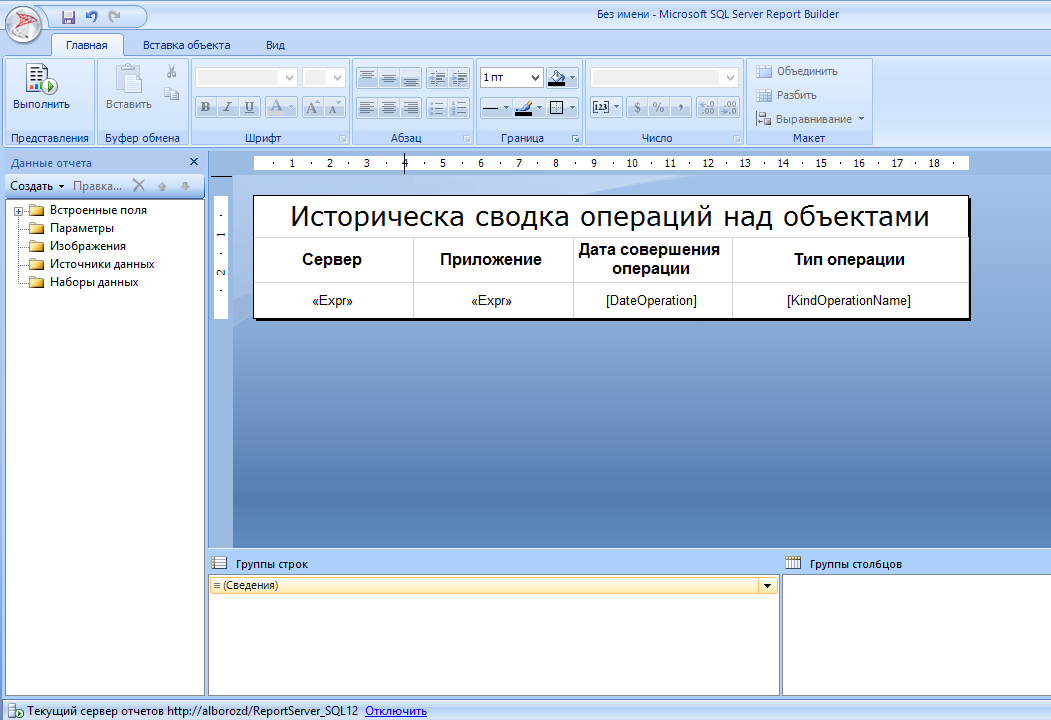


Рис 3.25. Интерфейс ReportBuilder

#### 3.9.4.2. Добавление отчетов в систему

Для добавления нового отчета необходимо перейти в раздел «Настройки» и пройти по ссылке «Настройка отчетов». В открывшемся окне, нужно заполнить все параметры добавляемого отчета (Рис 3.26).

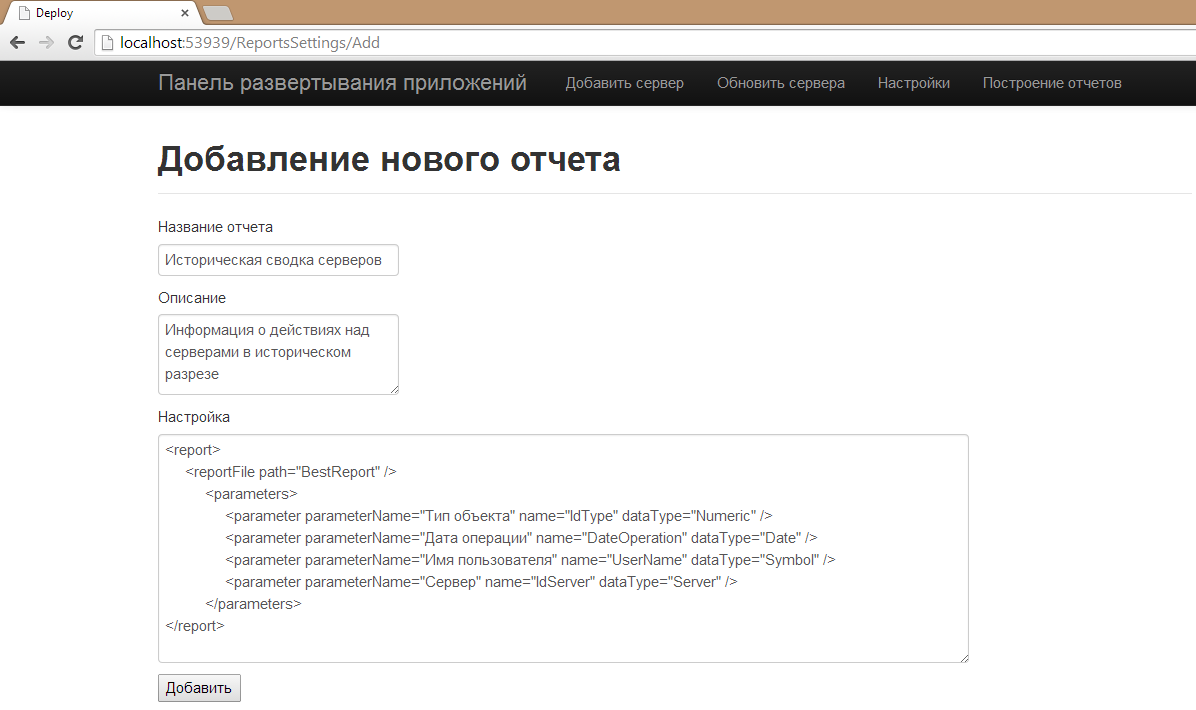


Рис 3.26. Добавление нового отчета

Поле Настройка отчета представляет собой xml текст, где корневым элементом должен быть узел в названием <report>. Элемент <reportFile> является обязательным, в нем указывается название файла отчета на сервере отчетов. Список элементов <parameters> не является обязательным. Эти элементы описывают параметры, которые принимает на вход отчет. Каждый элемент <parameter> должен иметь три атрибута: parameterName – отображаемое имя параметра, name – имя параметра внутри отчета, dataType – тип параметра. Тип параметра может принимать следующие значения:

* Date – параметр типа «Дата»;
* Numeric – числовой параметр;
* Symbol – символьный параметр;
* Server – объект сервера;
* DeployObject – объект сервера.

В зависимости от типа параметра будет генерироваться соответствующий контрол для выбора значения.

#### 3.9.4.3. Формирование отчетов

После добавления отчета в систему он появится в списке отчетов. Для начала формирования отчета, необходимо просто щелкнуть по нему и откроется окно ввода входных параметров, если они настроены в настройке отчета (Рис 3.27).

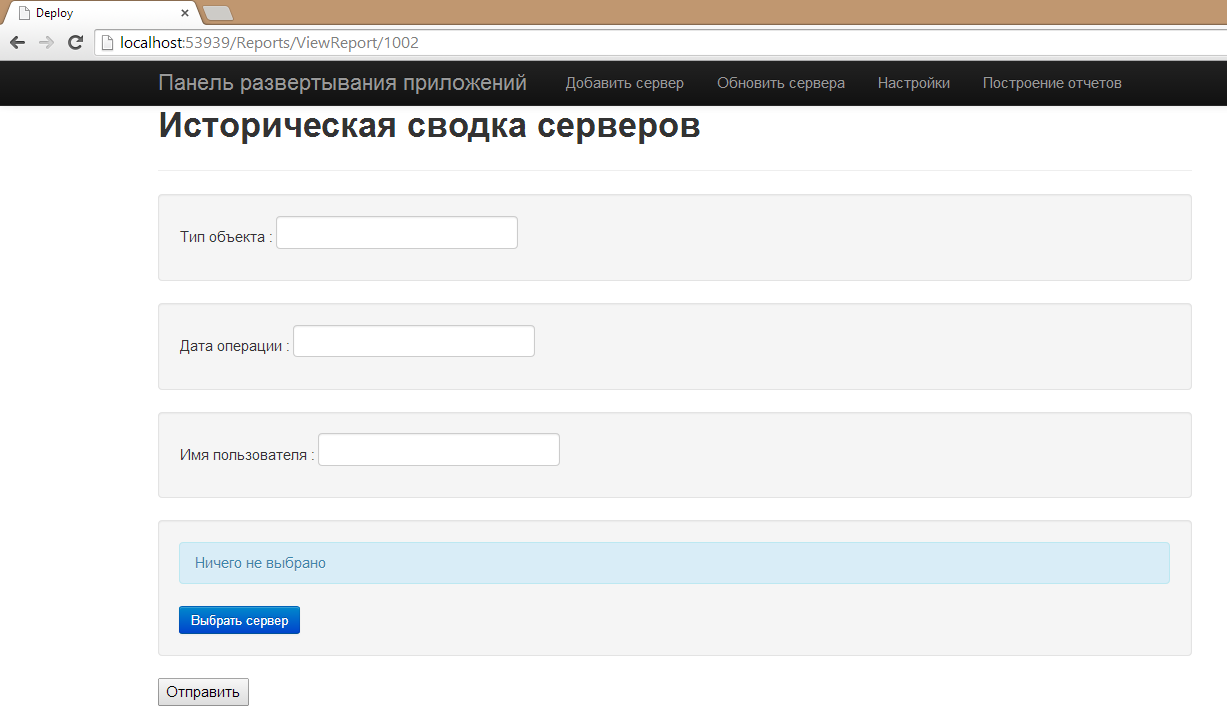


Рис 3.27. Окно ввода параметров отчета

После ввода всех необходимых параметров нужно нажать на кнопку отправить, и откроется новое окно, в котором будет сформирован отчет. На рисунке 3.28 показан сформированный отчет «Историческая сводка операций над объектом», который выводит информацию по всем объектам системы и операциям, которые совершались над данными объектами.

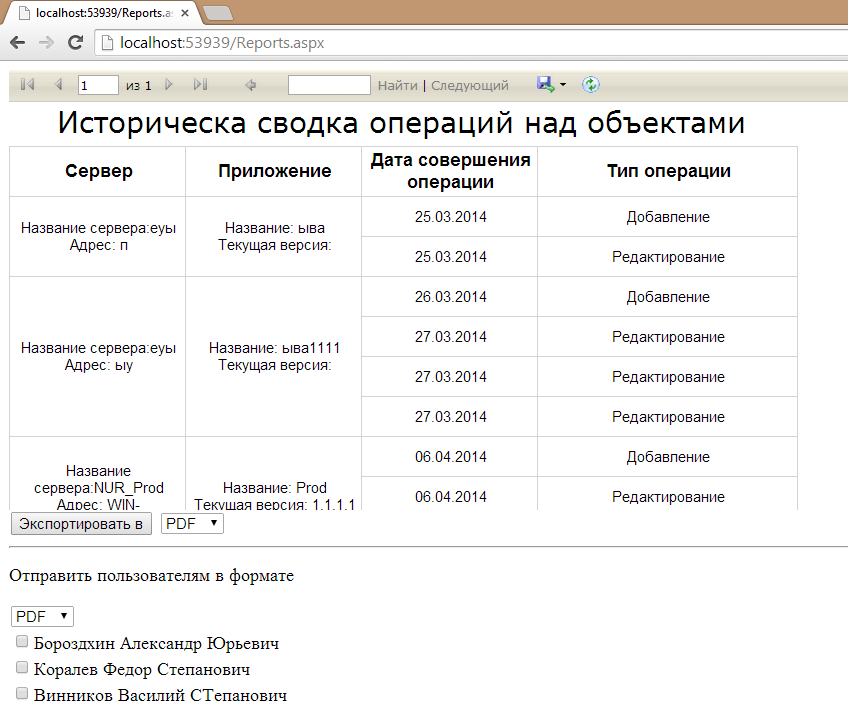


Рис. 3.28. Сформированный отчет» «Историческая сводка операций над объектом»

На рисунке 3.29 показан сформированный отчет «История обновлений приложения», который выводи информацию по всем операциям развертывания и обновления для конкретного объекта.

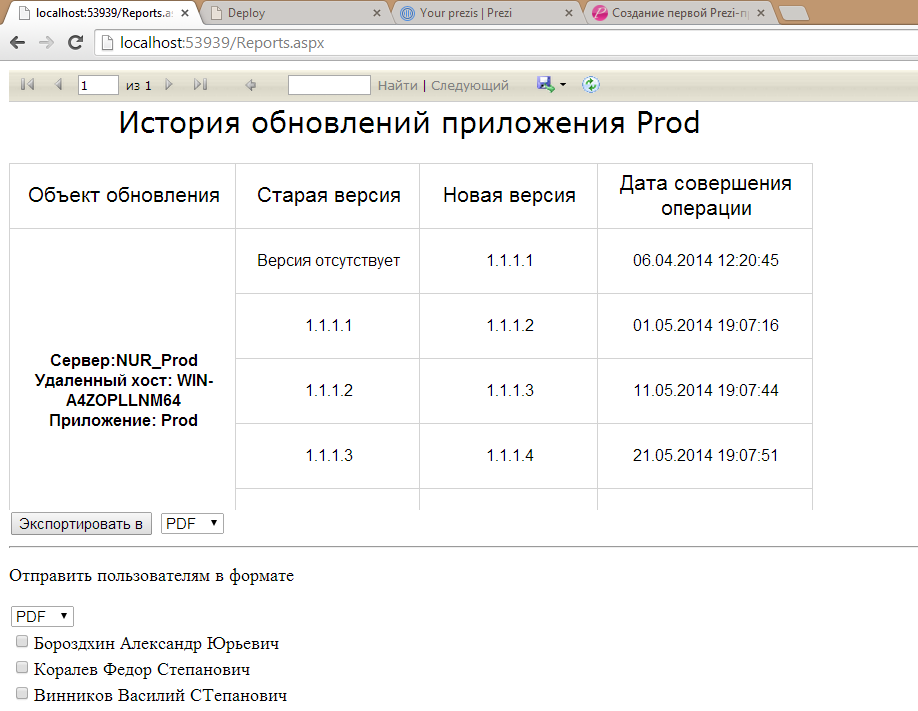


Рис 3.29. Сформированный отчет «История обновлений приложения»

На рисунке 3.30 показан сформированный отчет «История обновления объектов системы», который показывает информацию об обновлениях всех объектов с группировкой по серверам.

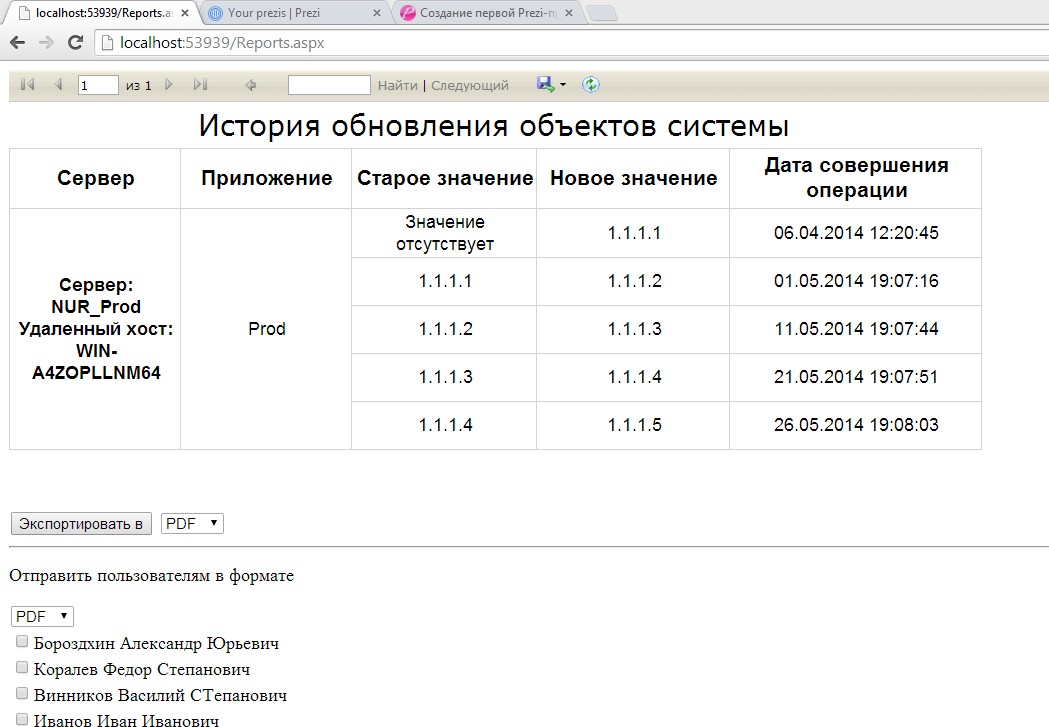


Рис 3.30. Сформированный отчет «История обновления объектов системы»

Сформированный отчет можно экспортировать в три формата: PDF, DOC, XLS. Так же есть возможность отправить сформированный отчет в выбранном формате на почту одному или нескольким пользователям.

## 3.10. Аппаратное и программное обеспечение, необходимое для внедрения разработанной системы

Для внедрения разработанной системы необходим сервер базы данных и сервер приложений. Рассмотрим требования к аппаратной части этих серверов более подробно:

***Технические характеристики сервера БД:***

- процессор – не менее 3 ГГц;

- объем оперативной памяти – не менее 1 Гб;

- дисковая подсистема – не менее 250 Гб;

- сетевой адаптер – FastEthernet 100;

- клавиатура – не менее 104 клавиш.

***Технические характеристики сервера приложений:***

- процессор – не менее 3 ГГц;

- объем оперативной памяти – не менее 1 Гб;

- дисковая подсистема – не менее 50 Гб;

- сетевой адаптер – FastEthernet 100;

- клавиатура – не менее 104 клавиш.

***Технические характеристики клиентских ПК:***

- процессор – не ниже P4 1500 МГц;

- объем оперативной памяти – не менее 512 Мб;

- дисковая подсистема – не менее 40 Гб;

- сетевой адаптер – FastEthernet 100;

- видеосистема – разрешающая способность не ниже 1024x768 точек;

- манипулятор типа «мышь»;

- клавиатура – не менее 104 клавиш (русифицированная).

# ЗАКЛЮЧЕНИЕ

В результате выполнения дипломной работы была разработана система «DOUIN» для централизованного хранения информации о продуктивных серверах и удаленного развертывания приложений. В процессе разработки программного обеспечения были пройдены все этапы проектирования от изучения предметной области до эксплуатации. Реализованная система отвечает всем предъявленным требованиям по функциональности и надежности.

Для разработки системы «DOUIN» было использовано современное средство разработки Visual Studio 2012 и язык программирования C# 5.0. В качестве технологии построения веб – приложений была выбрана ASP.NET MVC. Эта технология позволяет разрабатывать гибкие и легко масштабируемые приложения по архитектуре MVC. В качестве СУБД использована система управления базами данных MS Sql Server.

В ходе выполнения работы были применены знания по исследованию предметной области, анализу рынка программного обеспечения, концептуальному моделированию, проектированию баз данных и разработке программного обеспечения.

# СПИСОК ЛИТЕРАТУРЫ

1. Официальная страница Web Deployment Tool [Электронный ресурс]. –Электрон. текстовые да. - URL:http://www.iis.net/downloads/microsoft/web-deploy.
2. Официальный сайт IIS [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://www.iis.net/.
3. Мартин Р. Принципы, паттерны и методики гибкой разработки на языке C#. / Р. Мартин, М. Мартин – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 768  с.
4. Влюбляемся в Entity Framework [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://alexidsa.blogspot.ru/2009/03/entity-framework-1-ef.html.
5. Обучающие материалы SSRS [Электронный ресурс]. – Электрон. текстовые дан. – URL:http://msdn.microsoft.com/en-us/library/bb522859.aspx.
6. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. / Дж. Рихтер; 4-е изд. – СПБ.: Питер, 2014. – 896  с.
7. Станек У. Microsoft SQL Server 2012. Справочник администратора. / У. Станек; 3-е изд. – БХВ – Петербург, 2013. – 576 стр.
8. Сандерсон C. ASP.NET MVC Framework с примерами на C# для профессионалов. / С. Сандерсон; Пер. с англ. – М.: ООО «И.Д. Вильямс», 2010. – 560 с.
9. Основы архитектуры IIS, или запросопровод для ASP.NET [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://habrahabr.ru/post/189086/.
10. Официальный сайт jQuery [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://jquery.com/.
11. Официальный сайт Twitter Bootstrap [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://getbootstrap.com/.
12. Флэнаган Д. JavaScript. Подробное руководство. / Д. Флэнаган; Пер. с англ. – СПб: Символ\_Плюс, 2008. – 992 с.
13. Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. / Р. Мартин; – СПБ.: Питер, 2014. – 464 с.
14. Гамма Э. Приемы объектно – ориентированного программирования. Паттерны проектирования. / Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес; – СПБ.: Питер, 2014. – 368 с.
15. Макдональд М. Microsoft ASP.NET 4 с примерами на C# для профессионалов. / М. Макдональд, А. Фримен, М. Шпушта; 4-изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2011. – 1424 с.
16. Троэлсен Э. Язык программирования C# и платформа .NET 4.5. / Э.  Троэлсен; 6-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2013. – 1312  с.
17. Фримен Э. Паттерны проектирования. / Э. Фримен, Э. Фримен, К. Сьерра, Б. Бейтс; – СПБ.: Питер, 2011. – 656 с.
18. Хольцнер, С. jQuery. Практическое применение / С. Хольцнер. – М. : Эксмо, 2010. – 224 с.
19. Объектно-ориентированный анализ и проектирование [Электронный ресурс]. – Электрон. текстовые дан. – http://ooad.asf.ru.
20. Бибо Б. jQuery. Подробное руководство по продвинутому JavaScript./ Б.  Бибо, И. Кац. - СПб. : Символ-Плюс, 2008. - 624 с.
21. Хоп Г. Шаблоны корпоративных приложений / Г. Хоп; Вильямс, 2006. - 672 с.
22. Фаулер М. Рефакторинг. Улучшение существующего кода / М. Фаулер; СПб.: Символ-Плюс, 2008. - 432 с.
23. Шилдт Г. Полный справочник по C# / Г. Шилдт; 4-изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 740 с.
24. Джозеф С. LINQ: язык интегрированных запросов в C# 2008 для профессионалов. / С. Джозеф; Пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 560 с.
25. ASP.NET MVC Framework – ставим точки [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://habrahabr.ru/post/63138/.
26. Настоящее понимание ViewState'а [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://habrahabr.ru/post/119537/.
27. Обработка запросов IIS сервером в ASP.NET MVC [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://public-csharp.blogspot.ru/2010/10/iis-aspnet-mvc.html.
28. Кржиштоф Ц. Инфраструктура программных проектов. Соглашения, идиомы и шаблоны для многократно используемых библиотек .NET / Ц.  Кржиштоф, А. Брэд; Пер. с англ. – М.: ООО «И.Д. Вильямс», 2011. – 416  с.
29. Освоение Ajax: эффективный подход к созданию Web-сайтов [Электронный ресурс]. – Электрон. текстовые дан. – URL: https://www.ibm.com/developerworks/ru/library/wa-ajaxintro1/
30. Портал обучения HTML [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://htmlbook.ru/.
31. Официальный сайт ASP.NET [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://www.asp.net/.
32. Обучающие материалы по CSS и HTML [Электронный ресурс]. – Электрон. текстовые дан. – URL: http://www.teaching-materials.org/htmlcss/.

# ПРИЛОЖЕНИЕ 1. Реализация Базового репозитория.

public class BaseRepository<T> : IDao<T> where T : class

{

internal DbSet<T> dbSet;

DeployAppEntities \_dbModel;

public BaseRepository(DeployAppEntities dbModel)

{

\_dbModel = dbModel;

dbSet = dbModel.Set<T>();

}

public void Add(T entity)

{

if (entity == null)

throw new ArgumentNullException("entity");

dbSet.Add(entity);

}

public void Delete(int id)

{

var entity = dbSet.Find(id);

if (entity == null)

throw new InvalidOperationException("Произошла ошшибка при удалении объекта операции. Не найден объект с таким идентификатором.");

dbSet.Remove(entity);

}

public void Edit(T entity)

{

if (entity == null)

throw new ArgumentNullException("entity");

var pKey = dbSet.Create().GetType().GetProperty("id").GetValue(entity);

var oldEntity = dbSet.Find(pKey);

\_dbModel.Entry<T>(oldEntity).CurrentValues.SetValues(entity);

}

public IEnumerable<T> GetAll()

{

return dbSet.AsEnumerable<T>();

}

public T GetById(int id)

{

return dbSet.Find(id);

}

}

# ПРИЛОЖЕНИЕ 2. Реализация CRUD Контроллера

public abstract class CRUDController<T, M> : Controller where T : class

where M : class, IModel<T>

{

public CRUDController()

{

}

protected abstract BaseRepository<T> Repository { get; }

protected abstract DataManager Dm { get; }

public virtual ActionResult List(string search)

{

if (!string.IsNullOrEmpty(search))

return View(Repository.GetAll

return View(Repository.GetAll());

}

public ActionResult Add()

{

return View();

}

[HttpPost]

[ValidateInput(false)]

public ActionResult Add(M entity)

{

if (!ModelState.IsValid)

{

ViewData["errors"] = "Объект не валиден";

return View(entity);

}

Repository.Add(entity.ToEfEntity());

Dm.SaveChanges();

return RedirectToAction("List");

}

public ActionResult Edit(int id)

{

return View(Repository.GetById(id));

}

[HttpPost]

[ValidateInput(false)]

public ActionResult Edit(T entity)

{

if (!ModelState.IsValid)

{

ViewData["errors"] = "Объект не валиден";

return View(entity);

}

Repository.Edit(entity);

Dm.SaveChanges();

return RedirectToAction("List");

}

public ActionResult Delete(int id)

{

Repository.Delete(id);

Dm.SaveChanges();

return RedirectToAction("List");

}

public ActionResult Details(int id)

{return View(Repository.GetById(id));}}

# ПРИЛОЖЕНИЕ 3. Отсканированная страница Акта о внедрении ПО

