Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего профессионального образования

«Алтайский государственный технический университет им. И.И. Ползунова»

Факультет . информационных технологий

Кафедра . прикладная математика .

Курсовой проект защищен с оценкой \_\_\_\_\_\_\_\_ \_

Руководитель

проекта (работы) Старолетов М.С.

“\_\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_2013 г.

**Реализация репозитория для установленных программ в системе**

Пояснительная записка

к курсовой работе

по дисциплине Проектирование сетевых и многопоточных приложений

**КР 230105.07.000ПЗ**

Студент группы ПОВТ-81 Клиппа М.С.

Руководитель

проекта (работы) Старолетов С.М.

Барнаул 2013

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Задание на курсовую работу**  Учебная дисциплина: Проектирование сетевых и многопоточных приложений  ФИО студента: Клиппа Максим Сергеевич  Группа: ПОВТ-81  Задание: Реализация репозитория для установленных программ в системе.  Руководитель: Старолетов С. М. | | | | | | | | | | |
|  |  |  |  |  | **КР 230105.07.000ПЗ** | | | | | |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подпись | Дата |
| Разработал | | Клиппа М.С. |  |  | Реализация репозитория для установленных программ в системе | Лит. | | | Лист | Листов |
| Пров. | | Старолетов С.М. |  |  |  | У |  | 2 | 26 |
|  | |  |  |  | АлтГТУ, ФИТ  *гр. ПОВТ-81* | | | | |
| Н.контролер | |  |  |  |
| Утв. | | Кантор С.А. |  |  |

**Аннотация**

В данной работе представлены сведения о проектировании и разработке приложения "Репозиторий для установленных программ в системе". Репозиторий представляет собой набор [программного обеспечения](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5), позволяющего управлять процессом [установки](http://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%81%D1%82%D0%B0%D0%BB%D0%BB%D1%8F%D1%86%D0%B8%D1%8F_%28%D0%9F%D0%9E%29) и удаления различных компонентов программного обеспечения.

Приложение написано на языке программирования С# с использованием Microsoft Visual Studio 2010 и NET Framework 4.0. Взаимодействие серверной и клиентской части осуществляется при помощи программного [фреймворк](http://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)а WCF, делающего возможным построение безопасных и надёжных транзакционных систем через упрощённую унифицированную программную модель межплатформенного взаимодействия.

Исходный код приложения, а также данный отчет представлены на прилагаемом CD диске.

**Содержание**

[Введение 5](#_Toc345686078)

[1 Постановка задачи 6](#_Toc345686079)

[1.1 Развернутая постановка задачи 6](#_Toc345686080)

[2 Проектирование программного обеспечения 7](#_Toc345686081)

[2.1 Диаграмма вариантов использования 7](#_Toc345686082)

[2.2 Диаграмма развертывания 8](#_Toc345686083)

[2.3 Временная диаграмма 8](#_Toc345686084)

[2.4 Диаграмма деятельности 9](#_Toc345686085)

[2.5 Диаграмма классов UML 10](#_Toc345686085)

[3 Общий подход к реализации программного обеспечения 11](#_Toc345686086)

[Заключение 12](#_Toc345686089)

[Список использованных источников 13](#_Toc345686090)

[Приложение А Руководство пользователя 14](#_Toc345686091)

[Приложение Б Текст программы 15](#_Toc345686093)

**Введение**

Современные системы состоят из огромного числа зависимых библиотек, исполняемых файлов, скриптов и т.д. Удаление или изменение версии одного из составляющих систему компонентов может повлечь неработоспособность других, связанных с ним компонентов, или даже вывести из строя всю систему. Для поддержания целостности и обеспечения возможности распространения программ используются менеджеры пакетов. Пакетсодержит собранную программу, информацию о том, какие требуется совершить действия для ее установки, информацию о зависимостях**,** а также возможно много других данных, в зависимости от вида пакета. Часто компоненты, используемые различными программами, выделяют в отдельные пакеты и помечают, что для работы ПО, предоставленного пакетом A, необходимо установить пакет B. В таком случае говорят, что пакет A зависитот пакета B или что между пакетами A и B существует зависимость. Отслеживание зависимостей между такими пакетами представляет собой серьёзную задачу для любого дистрибутива— некоторые компоненты могут быть взаимозаменяемыми и при удовлетворении тех или иных требований может обнаружиться несколько пакетов, предлагающих затребованный ресурс. Контроль целостности и непротиворечивости установленного в системе ПО представляет собой еще более сложную задачу.

Менеджер пакетов (репозиторий) - это набор инструментов, позволяющих автоматизировать процесс установки, обновления, конфигурирования и удаления пакетов программного обеспечения с компьютера.

# 1 Постановка задачи

## 1.1 Развернутая постановка задачи

Спроектировать и реализовать репозиторий для установленных программ в

системе.

Аналог менеджера пакетов в Linux. Должен позволять скачать с сервера некое программное обеспечение (в виде набора файлов), устанавливать их в выбранное место, притом устанавливать другие пакеты, от которых текущий пакет зависит. Также должен позволять удалять пакеты из системы.

# 2 Проектирование программного обеспечения

## 2.1 Диаграмма вариантов использования

На рисунке 2.1 представлена диаграмма использования для нашего приложения.

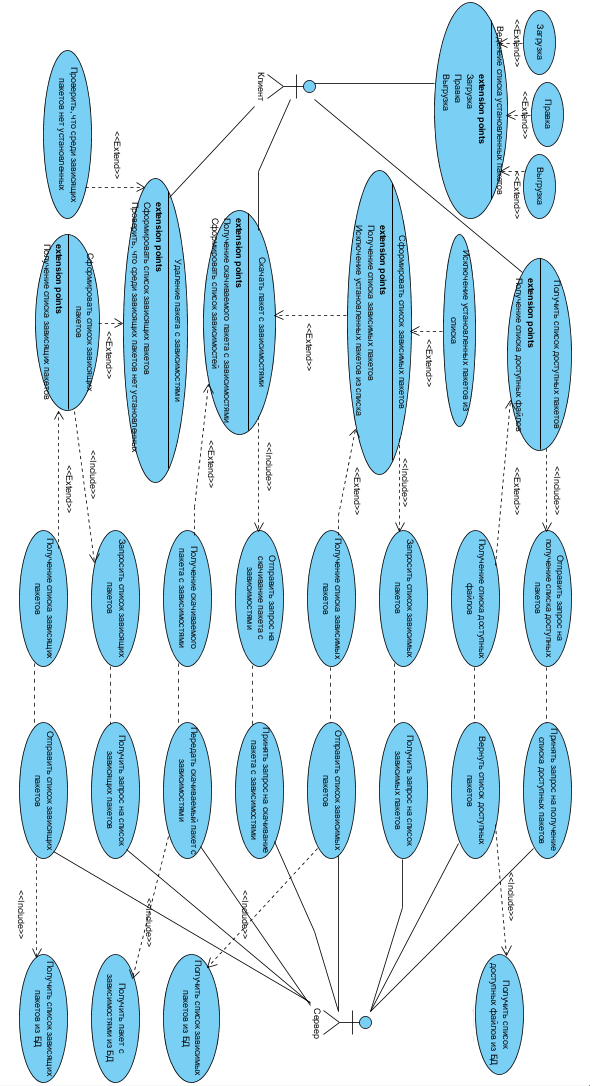


Рисунок 2.1 - Диаграмма вариантов использования

## 2.2 Диаграмма развертывания

На рисунке 2.2 представлена диаграмма развертывания приложения. В ней отражены необходимые файлы и библиотеки для функционирования программы.

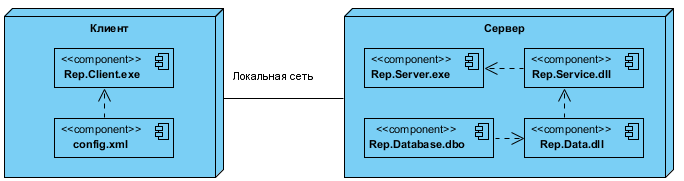


Рисунок 2.2 - Диаграмма развертывания

## 2.3 Временная диаграмма

На рисунке 2.3 представлена временная диаграмма. Данная диаграмма отображает наиболее полный вариант работы программы.

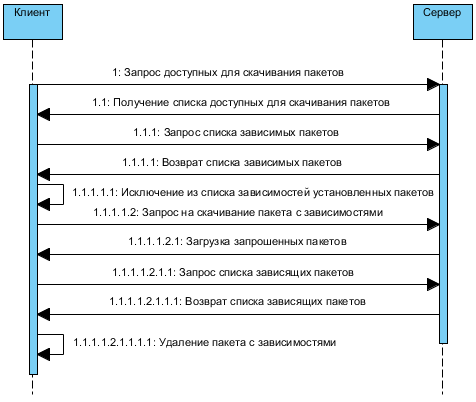


Рисунок 2.3 - Временная диаграмма

## 2.4 Диаграмма деятельности

На рисунке 2.4 представлена диаграмма деятельности.

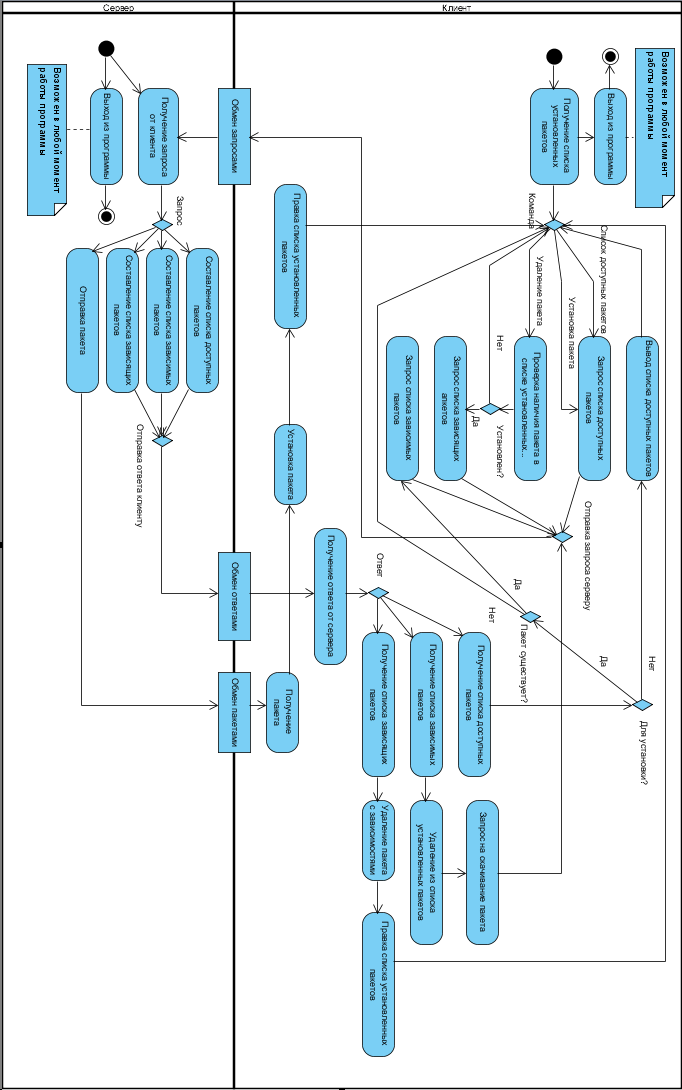
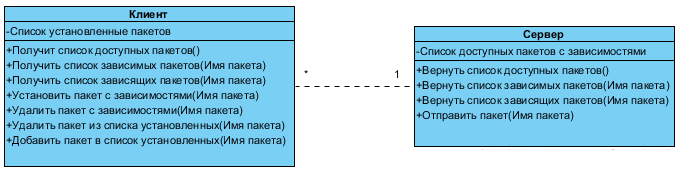


Рисунок 2.4 - Диаграмма деятельности

**2.5 Диаграмму классов UML**

На рисунке 3.1 приведена диаграмма концептуальных классов приложения, их атрибуты, методы и взаимосвязи.



# Рисунок 2.5 - Диаграмма концептуальных классов

## 3 Общий подход к реализации программного обеспечения

Данный программный продукт представляет собой клиент-серверное приложение, написанное с использованием библиотеки .NET Framework 4.0 на языке C#. Взаимодействие серверной и клиентской части осуществляется при помощи программного [фреймворк](http://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)а WCF, делающего возможным построение безопасных и надёжных транзакционных систем через упрощённую унифицированную программную модель межплатформенного взаимодействия. Приложение было написано в среде Microsoft Visual Studio 2010 и протестировано в операционной системе Windows 7 32 bit.

Приложение было разделено на две части:

* серверная часть;
* клиентская часть;

Серверная часть служит хостингом для службы WCF, являющейся посредником при обмене данными между клиентом и базой данных.

Клиентская часть приложения предоставляет конечному пользователю следующий набор функций:

* получение списка доступных для установки пакетов;
* установка пакета с зависимостями;
* удаление пакета с зависимостями (каждый пакет удаляется в отдельном потоке).

C# — язык программирования, сочетающий объектно-ориентированные и контекстно-ориентированные концепции. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML, поэтому он был выбран для реализации приложения.

# Заключение

В результате работы над проектом были изучены возможности организации работы приложений по сети на языке программирования C# с использованием WCF службы. Также изучены возможности языка для реализации многопоточных приложений.

Все основные требования к приложению были реализованы.

# Список использованных источников

1. Учебник по началу работы с WCF [Электронный ресурс] Режим доступа: http://msdn.microsoft.com/ru-ru/library/ms734712.aspx
2. Размещение и использование служб WCF [Электронный ресурс] Режим доступа: http://msdn.microsoft.com/ru-ru/library/bb332338.aspx
3. Отображение прогресса загрузки данных в сервисах WCF [Электронный ресурс] Режим доступа: http://www.gotdotnet.ru/blogs/sergun/6702/

# Приложение А

# Руководство пользователя

Серверная часть приложения служит хостингом для службы WCF и не обременена никаким дополнительным функционалом. По существу сервер является консольным приложением, выход из которого осуществляется по нажатию произвольной клавиши на клавиатуре.

Клиентская часть также реализована в виде консольного приложения, имеющего в своем арсенале следующий набор команд:

* ? – получение справки по работе приложения;
* get – получение списка доступных для скачивания пакетов;
* install [*package\_name*] – установка пакета с соответствующим именем, включая зависимые пакеты;
* delete [*package\_name*] – удаление пакета с соответствующим именем, включая зависимые пакеты;
* exit – выход из программы.

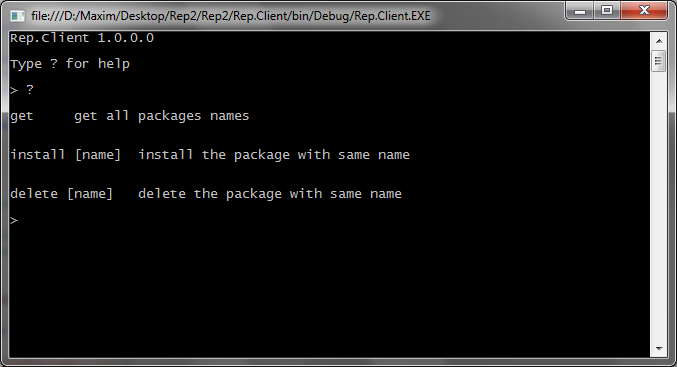


Рисунок А.1 – Справка по работе приложения

# Приложение Б

# Текст программы

**Клиент**

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using Rep.Client.RepService;

using System.Windows.Forms;

using System.Xml.Serialization;

using System.IO;

using System.Threading;

namespace Rep.Client

{

static class Program

{

static XmlSerializer serializer;

static List<PackageInfo> installedPackages;

[STAThread]

static void Main()

{

serializer = new XmlSerializer(typeof(List<PackageInfo>));

LoadInstalledPackagesList();

var assembly = Assembly.GetExecutingAssembly().GetName();

Console.WriteLine("{0} {1}\n", assembly.Name, assembly.Version);

Console.WriteLine("Type ? for help\n");

Console.Write("> ");

while (Process(Console.ReadLine())) { Console.Write("> "); }

}

private static void LoadInstalledPackagesList()

{

installedPackages = new List<PackageInfo>();

try

{

FileStream fs = new FileStream("config.xml", FileMode.Open);

installedPackages = (List<PackageInfo>)serializer.Deserialize(fs);

fs.Close();

}

catch (System.Exception ex)

{

}

}

private static void UnloadInstalledPackagesList()

{

TextWriter writer = new StreamWriter("config.xml");

serializer.Serialize(writer, installedPackages);

writer.Close();

}

private static void ThreadProc(object packageName)

{

File.Delete(packageName.ToString());

}

static bool Process(string command)

{

if (command.ToLower() == "exit")

return false;

if (command.ToLower() == "?")

{

Console.WriteLine("\nget\tget all packages names\n");

Console.WriteLine("\ninstall [name]\tinstall the package with same name\n");

Console.WriteLine("\ndelete [name]\tdelete the package with same name\n");

} else

if (command.ToLower() == "get")

{

using (var client = new RepServiceClient())

{

PackageInfo[] packages = client.GetPackages();

Console.WriteLine("\nPackages:\n");

foreach (var packageInfo in packages)

{

Console.WriteLine("{0}\t{1}", packageInfo.Name, packageInfo.Version);

}

Console.WriteLine();

}

}

else

{

string packageName = command.ToLower().Split(' ')[1];

command = command.Split(' ')[0];

int packageID = -1;

if (command.ToLower() == "install")

{

packageID = Download(packageName, packageID);

}

if (command.ToLower() == "delete")

{

packageID = Delete(packageName, packageID);

}

}

return true;

}

private static int Delete(string packageName, int packageID)

{

if (PackageIsInstalled(packageName, true))

{

using (var client = new RepServiceClient())

{

packageID = GetPackageID(client.GetPackages(), packageName);

var installedPackagesID = GetInstalledPackagesID();

var deletingPackagesID = GetDeletingPackagesID(client, packageID);

var deletingPackagesName = new List<string>();

foreach (var deletingPackageID in deletingPackagesID)

{

if (!PackageIsDependent(installedPackagesID, deletingPackageID, client))

{

//File.Delete(client.GetPackages().FirstOrDefault(p => p.Id == deletingPackageID).Name + ".7z");

deletingPackagesName.Add(client.GetPackages().FirstOrDefault(p => p.Id == deletingPackageID).Name + ".7z");

RemoveDeletedPackFromList(deletingPackageID);

installedPackagesID.Remove(deletingPackageID);

}

//else break;

}

ThreadingDeleting(deletingPackagesName);

}

Console.WriteLine();

UnloadInstalledPackagesList();

} return packageID;

}

private static void ThreadingDeleting(List<string> deletingPackagesName)

{

List<Thread> threads = new List<Thread>();

foreach (var deletingPackageName in deletingPackagesName)

{

threads.Add(new Thread(ThreadProc));

}

for (int i = 0; i < deletingPackagesName.Count; i++)

{

threads[i].Start(deletingPackagesName[i]);

}

}

private static void RemoveDeletedPackFromList(int deletingPackageID)

{

for (int i = 0; i < installedPackages.Count; i++)

{

if (installedPackages[i].Id == deletingPackageID)

{

installedPackages.RemoveAt(i);

break;

}

}

}

private static int Download(string packageName, int packageID)

{

if (!PackageIsInstalled(packageName))

{

using (var client = new RepServiceClient())

{

PackageInfo[] packages = client.GetPackages();

if (PackageIsExist(packages, packageName))

{

packageID = GetPackageID(packages, packageName);

var needPackagesID = client.GetDependencyStack(packageID).ToList();

needPackagesID.Reverse();

var installedPackagesID = GetInstalledPackagesID();

foreach (int needPackageID in needPackagesID)

{

if (!installedPackagesID.Contains(needPackageID))

{

DownloadPackage(client, packages, needPackageID);

Console.WriteLine();

}

}

DownloadPackage(client, packages, packageID);

Console.WriteLine("\nDownload is successfully completed\n");

UnloadInstalledPackagesList();

}

}

} return packageID;

}

private static bool PackageIsDependent(List<int> installedPackagesID, int deletingPackageID, RepServiceClient client)

{

int[] upPackagesID = client.GetUpPackagesStack(deletingPackageID);

string packageName = client.GetPackages().FirstOrDefault(p => p.Id == deletingPackageID).Name;

foreach (var upPackageID in upPackagesID)

{

if (installedPackagesID.Contains(upPackageID))

{

Console.WriteLine("Package \"" + packageName + "\" can't be deleted. It's a dependent package.");

return true;

}

}

Console.WriteLine("Package \"" + packageName + "\" is successfully deleted.");

return false;

}

private static List<int> GetDeletingPackagesID(RepServiceClient client, int packageID)

{

var deletingPackagesID = client.GetDependencyStack(packageID).ToList();

deletingPackagesID.Insert(0, packageID);

return deletingPackagesID;

}

private static int GetPackageID(PackageInfo[] packages, string packageName)

{

return packages.FirstOrDefault(p => p.Name == packageName).Id;

}

private static void DownloadPackage(RepServiceClient client, PackageInfo[] packages, int downloadingPackageID)

{

Downloading(client, downloadingPackageID, packages.FirstOrDefault(p => p.Id == downloadingPackageID).Name);

installedPackages.Add(packages.FirstOrDefault(p => p.Id == downloadingPackageID));

}

private static List<int> GetInstalledPackagesID()

{

List<int> installedPackagesID = new List<int>();

foreach (var pack in installedPackages)

{

installedPackagesID.Add(pack.Id);

}

return installedPackagesID;

}

private static bool PackageIsExist(PackageInfo[] packages, string packageName)

{

foreach (var pack in packages)

{

if (pack.Name == packageName)

{

return true;

}

}

Console.WriteLine("\nThis package not exist\n");

return false;

}

private static bool PackageIsInstalled(string packageName, bool deletingFlag = false)

{

foreach (var pack in installedPackages)

{

if (packageName == pack.Name)

{

if (!deletingFlag) Console.WriteLine("\nThis package is installed\n");

return true;

}

}

if (deletingFlag) Console.WriteLine("\nThis package is not installed\n");

return false;

}

private static void Downloading(RepServiceClient client, int id, string fileName)

{

Stream inputStream;

long length = client.DownloadPackage(id, out inputStream);

using (var writeStream = new FileStream(fileName + ".7z", FileMode.CreateNew, FileAccess.Write))

{

const int bufferSize = 16777216;

var buffer = new byte[bufferSize];

do

{

int bytesRead = inputStream.Read(buffer, 0, bufferSize);

if (bytesRead == 0)

{

break;

}

writeStream.Write(buffer, 0, bytesRead);

Console.Write("\r Installing " + fileName + " " + (writeStream.Position \* 100 / length) + "%");

}

while (true);

writeStream.Close();

}

inputStream.Dispose();

}

}

}

**App.config**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="BasicHttpBinding\_IRepService" closeTimeout="00:01:00"

openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"

allowCookies="false" bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard"

maxBufferSize="65536" maxBufferPoolSize="524288" maxReceivedMessageSize="20134217728"

messageEncoding="Mtom" textEncoding="utf-8" transferMode="Streamed"

useDefaultWebProxy="true">

<readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"

maxBytesPerRead="4096" maxNameTableCharCount="16384" />

<security mode="None">

<transport clientCredentialType="None" proxyCredentialType="None"

realm="" />

<message clientCredentialType="UserName" algorithmSuite="Default" />

</security>

</binding>

</basicHttpBinding>

</bindings>

<client>

<endpoint address="http://localhost:7777/rep" binding="basicHttpBinding"

bindingConfiguration="BasicHttpBinding\_IRepService" contract="RepService.IRepService"

name="BasicHttpBinding\_IRepService" />

</client>

</system.serviceModel>

</configuration>

**Сервер**

**Rep.Data**

**App.config**

<?xml version="1.0" encoding="utf-8"?>

<configuration>

<connectionStrings>

<add name="RepContext" connectionString="metadata=res://\*/RepModel.csdl|res://\*/RepModel.ssdl|res://\*/RepModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=.\sqlexpress;initial catalog=Rep.Database;integrated security=True;pooling=False;multipleactiveresultsets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

</configuration>

**Rep.Service**

**PackageInfo.cs**

using System.Runtime.Serialization;

namespace Rep.Service

{

[DataContract]

public class PackageInfo

{

[DataMember]

public int Id { get; set; }

[DataMember]

public string Name { get; set; }

[DataMember]

public string Version { get; set; }

}

}

**RemoteInfo.cs**

using System;

using System.IO;

using System.ServiceModel;

namespace Rep.Service

{

[MessageContract]

public class RemoteInfo : IDisposable

{

[MessageHeader(MustUnderstand = true)]

public string FileName { get; set; }

[MessageHeader(MustUnderstand = true)]

public long Length { get; set; }

[MessageBodyMember(Order = 1)]

public Stream FileByteStream { get; set; }

public void Dispose()

{

if (FileByteStream != null)

{

FileByteStream.Close();

FileByteStream = null;

}

}

}

[MessageContract]

public class DownloadRequest

{

[MessageBodyMember]

public int Id { get; set; }

}

}

**IRepService.cs**

using System.ServiceModel;

using System.Collections.Generic;

namespace Rep.Service

{

[ServiceContract]

public interface IRepService

{

[OperationContract]

PackageInfo[] GetPackages();

[OperationContract]

RemoteInfo DownloadPackage(DownloadRequest request);

[OperationContract]

int[] GetDependencyStack(int id);

[OperationContract]

int[] GetUpPackagesStack(int id);

}

}

**RepService.svc**

using System;

using System.IO;

using System.Linq;

using Rep.Data;

using System.Collections.Generic;

namespace Rep.Service

{

public class RepService : IRepService

{

public PackageInfo[] GetPackages()

{

using (var context = new RepContext())

{

return context.Packages

.Select

(p => new PackageInfo

{

Id = p.Id,

Name = p.Name,

Version = p.Version

})

.ToArray();

}

}

public RemoteInfo DownloadPackage(DownloadRequest request)

{

Package package;

using (var context = new RepContext())

{

package = context.Packages.FirstOrDefault(p => p.Id == request.Id);

}

if (package == null)

throw new Exception("Package not found.");

var fileInfo = new FileInfo(package.FullPath);

if (fileInfo.Exists == false)

throw new Exception("Package not found.");

var stream = new FileStream(package.FullPath, FileMode.Open, FileAccess.Read);

var result = new RemoteInfo

{

FileName = fileInfo.Name,

Length = fileInfo.Length,

FileByteStream = stream

};

return result;

}

public List<int> GetDependencyStack(List<int> stack, int id)

{

Package package;

using (var context = new RepContext())

{

package = context.Packages.FirstOrDefault(p => p.Id == id);

foreach (var pack in package.NeedPackages.ToArray())

{

stack.Add(pack.Id);

GetDependencyStack(stack, pack.Id);

}

return stack;

}

}

public int[] GetDependencyStack(int id)

{

return GetDependencyStack(new List<int>(), id).ToArray();

}

public int[] GetUpPackagesStack(int id)

{

List<int> packagesID = new List<int>();

using (var context = new RepContext())

{

foreach (var packages in context.Packages.FirstOrDefault(p => p.Id == id).Packages)

{

packagesID.Add(packages.Id);

}

}

return packagesID.ToArray();

}

}

}

**WebConfig.cs**

<?xml version="1.0"?>

<configuration>

<connectionStrings>

<add name="RepContext" connectionString="metadata=res://\*/RepModel.csdl|res://\*/RepModel.ssdl|res://\*/RepModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=.\sqlexpress;initial catalog=Rep.Database;integrated security=True;pooling=False;multipleactiveresultsets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

<system.web>

<compilation debug="true" targetFramework="4.0" />

</system.web>

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="FileTransferServicesBinding" transferMode="Streamed" messageEncoding="Mtom" maxReceivedMessageSize="20134217728" />

</basicHttpBinding>

</bindings>

<services>

<service behaviorConfiguration="RepService" name="FileService.FileTransferService">

<endpoint address="getfile"

binding="basicHttpBinding"

bindingConfiguration="FileTransferServicesBinding"

contract="FileService.IFileTransferService" />

<host>

<baseAddresses>

<add baseAddress="http://localhost/" />

</baseAddresses>

</host>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="RepService">

<!-- To avoid disclosing metadata information, set the value below to false and remove the metadata endpoint above before deployment -->

<serviceMetadata httpGetEnabled="true"/>

<!-- To receive exception details in faults for debugging purposes, set the value below to true. Set to false before deployment to avoid disclosing exception information -->

<serviceDebug includeExceptionDetailInFaults="false"/>

</behavior>

</serviceBehaviors>

</behaviors>

<serviceHostingEnvironment multipleSiteBindingsEnabled="true" />

</system.serviceModel>

<system.webServer>

<modules runAllManagedModulesForAllRequests="true"/>

</system.webServer>

</configuration>

**Rep.Server**

**Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.ServiceModel;

using System.Text;

namespace Rep.Server

{

class Program

{

static void Main()

{

var myServiceHost = new ServiceHost(typeof(Service.RepService));

myServiceHost.Open();

Console.ReadKey();

myServiceHost.Close();

}

}

}

**App.config**

<?xml version="1.0" encoding="utf-8" ?>

<configuration>

<connectionStrings>

<add name="RepContext" connectionString="metadata=res://\*/RepModel.csdl|res://\*/RepModel.ssdl|res://\*/RepModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=.\sqlexpress;initial catalog=Rep.Database;integrated security=True;pooling=False;multipleactiveresultsets=True;App=EntityFramework&quot;" providerName="System.Data.EntityClient" />

</connectionStrings>

<system.serviceModel>

<bindings>

<basicHttpBinding>

<binding name="FileTransferServicesBinding" transferMode="Streamed" messageEncoding="Mtom" maxReceivedMessageSize="20134217728" />

</basicHttpBinding>

</bindings>

<services>

<service behaviorConfiguration="MyServiceTypeBehaviors" name="Rep.Service.RepService">

<endpoint address="rep"

binding="basicHttpBinding"

bindingConfiguration="FileTransferServicesBinding"

contract="Rep.Service.IRepService" />

<host>

<baseAddresses>

<add baseAddress="http://localhost:7777/" />

</baseAddresses>

</host>

</service>

</services>

<behaviors>

<serviceBehaviors>

<behavior name="MyServiceTypeBehaviors">

<serviceMetadata httpGetEnabled="true" />

<serviceDebug includeExceptionDetailInFaults="true" />

</behavior>

</serviceBehaviors>

</behaviors>

</system.serviceModel>

</configuration>