Dawit Nida

Developing Interoperable Online Backup Software

Helsinki Metropolia University of Applied Sciences
Bachelor of Engineering
Information Technology
Bachelor’s Thesis
25 November 2011

Author(s) Title Number of Pages Date	Dawit Nida Developing interoperable online backup software 38 pages + 7 appendices 25 November 2011
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kari Aaltonen, Principal Lecturer Jussi Hirvi, Project Supervisor and Manager

With ever-increasing amounts of digital data, various data storing techniques can be applied to overcome and minimize the risk of losing a single file or the whole system data. Data can be stored using different mechanisms including online backup. The main objective of this project was to design and implement interoperable online backup software initiated by the Green Spot Media Farm company residing in Helsinki, Finland. In addition, this documentation focuses on establishing a fundamental background and research to create a cross platform application using Windows Communication Foundation (WCF) (Microsoft's .NET Framework) on a Linux platform.

The software was developed using the Mono .NET development framework and Linux shell scripting based on a standard software development life cycle. This application uses a client-server paradigm in which suitable compliers, MySQL and Apache, programs were installed and configured. To achieve multiple users who backup simultaneously, an asynchronous communication method was implemented. Furthermore, it was deployed and tested on a Windows 7 platform.

As a result, multiple clients were allowed to access the host, create new backups and store data on the remote server concurrently. Besides, the application was analyzed using the Mono Migration Analyzer (MoMA) tool for porting to Linux. Hence, some of the class libraries were found missing. The application can be further developed to support different platforms, such as smart phones, tablets, MacOSx, Linux distribution and also to make new provision and customization to build a web-based application. Moreover, it can be optimized by allocating bandwidth limit and upgraded to provide additional features.

Keywords	Backup, Asynchronous Socket, Network Stream, C#, Bash, WCF, Mono

Contents

Acronyms

AOT	Ahead-of-Time Compiler
API	Application Programming Interface
BSD	Berkeley Software Distribution
CentOS	Community Enterprise Operating System
CLI	Common Language Runtime Compilation
DLL	Data Definition Language
DSA	Digital Signature Algorithm
EXE	Executable Files
FTP	File Transfer Protocol
GUI	Graphical User Interface
HLP	Help Files
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
INI	Initialization/Configuration File
JIT	Just-in-Time
MAC	Message Authentication Code
MoMA	Mono Migration Analyzer
OS	Operating System
P2P	Peer-to-Peer
PDA	Personal Digital Assistant
RSA	Rivest, Shamir and Adleman
RSH	Remote Shell
SSL	Secure Socket Layer
SYS	System Files
TCP	Transmission Control Protocol
TMP	Temporary Files
UDP	User Datagram Protocol
UML	Unified Modeling Language
WCF	Windows Communication Foundation
Winsock	Windows Sockets
XML	Extensible Markup Language

1 Introduction

Throughout the course of years, the amount of computer-based data has increased exponentially regardless of all the constraints and drawbacks. Losing a single file or the whole system data from a personal or an enterprise workstation can cause devastation to any kind of personal, business or service oriented company. Thus, backing up data to a safe place and protecting them from any software or hardware failure is considered as prolonging the companies' existence despite all the compulsory factors for in a centralized or distributed system. Different techniques can be applied to backup and restore one's essential and critical data.

An online data backup system is merely for storing or makes a second copy of computer data on another device or a remote host for safety and re-usability. Hence, the essence of online backup to a remote host persistently is taken as a more effective solution than the customary way. An online data backup is considered as a handy and efficient way on the basis of easiness, security, scalability, privacy, and various other features. Due to these reasons, the majority of small businesses or large companies and private customers are commencing to use online backup services.

Due to a growing demand on these services, this project was proposed and supplemented by Green Spot Media Farm Company, which resides in Helsinki, Finland. The objective of this final year project is to design and implement interoperable online data backup software for clients. In depth, software design procedures, implementation and deployment with integrated testing of the software are explained in this document. The document includes the overall functionalities and practicalities of the application from both the user and administrative perspective. In addition to these, it also contains concise analysis and discusses considerable research on major focal points on creating cross-platform (interoperable) application between Microsoft .NET Framework, in particular Windows Communication Foundation (WCF) technology, and the Linux platform.

The application uses Community Enterprise Operating System (CentOS), a free operating system (OS) based on Red Hat Enterprise Linux, as a back-end to store clients' data and an installable Windows-based desktop application for front-end users. The application utilizes and implements an asynchronous socket for handling multiple clients on-demand and implements an asynchronous network streaming technique to transfer data across the network.

2 Overall Description of Online Backup Software

2.1 Types of Backup

In a modest sense, backuping up means creating a secondary copy of a computer’s data and making an archive or duplicate on a local hard drive, removable disk or remote server to protect a user from data loss, which may occur due to inadvertent action, system corruption, hardware failure, natural disasters or malicious attacks. Currently, traditional backup methods such as optical disks, external hard disks, and USB devices are considered to be old-fashioned, costly and unreliable. Thus technology experts introduced a new technique for backing up, known as the online backup system.

An online backup system is used to manage bi-directional file transferring to/from a client’s computer from/to an off-site storage media via a secure Internet connection without any user intervention. Depending on backup capabilities and media used, backups can be of different kinds; there is, for example, individual file backup, individual folder backup, entire system backup (image backup), file-in-use backup, registry backup, database backup, network backup, and dump backup. [1, 2.] Depending on the type of technique used, backup and the restore methods can also be categorized into three as follows: [3].

Full Backup (Reference Backup) is a mechanism in which the entire system including applications and folders (directories) are included in the backup set and stored on the backup media. Time span to execute the backup of each single file and folder, storage space and network consistency can be considered as downsides for this kind of backup. On the other hand, restoring time and single backup set restoring are regarded as advantages.

Incremental Backup is another way to backup that is used to backup explicitly changed files after the most up-to-date full backup. In an incremental backup, all files in the folders existing inside a top folder are included automatically. Some of the benefits of incremental backup are that it is fast, it requires less data storage space, and files with a similar name can be stored as numerous subversions, but restoring data takes longer than with a full backup.

Differential Backup is the third type of backup used to store copies of newly added and changed files since the last full backup. Reduced backup time and a smaller amount of disk space are required than with full or incremental backups. However, restoring all files may take a significantly longer time since both, the last differential and full backup, have to be done simultaneously.

2.2 Features of Online Data Backup

An online data backup can be built as a standard client application or as a web-based application. Unlike their unique behaviors in implementation and deployment, both share a common feature in a real-world implementation. Performance can be mentioned as the prime difference between standard Windows application and web-based application. A standalone Windows application running in the background affects the performance of a computer compared to a web-based one. Some of the substantial features are summarized in table 1.

Table 1. Features of online data backup software

Fully Automatic	Users can make a backup or restore continuously in the background without any interference after setup rules are applied.
Multicomputer	This feature is used by clients with a centralized system having several computers which need to access the service using a single account without any prerequisites using the same application interface.
Accessibility	This feature refers to online accessing of backup service from anywhere using PCs, Personal Digital Assistants (PDAs), mobile phones, or tablets.
Search Engine	The search engine in an online backup system is used to support file enquiries as an internal search method on the local hard disk or the remote service provider.
Security Measures	Besides authorizing clients to access their own data, data transfer must be given a high priority for secure data streaming between the client and the service provider from intrusion or any kind of security risks.
Data Encryption	This feature describes transforming plaintext files into cypher text using hash-tables and keys to create non-readable data to unauthorized parties while transferring/streaming data over the network.

Data encryption, security, fully automatic features, and creating a schedule for the backup and restore processes are considered as the core features of any online data backup software. As summerized in table 1, data synchronization with different platforms, the web interface or the client's standalone application can have the extensibility for successful data management between various interfaces, the cloud, tablets and mobile phones to achieve a reliable and dependable system.

2.3 Existing System Overview

Although there are limited numbers of online data backup software vendors on the market, the existing ones provide different features and qualities. According to an online data backup review, as shown in table 2, SugarSync has the highest overall rating having various backup features and tools. Dropbox, which is currently competing to attract several clients, is easy to use and contains a different variety of tools and options for computer and mobile usage. Dropbox is not only used to backup files and folders online but also to synch files with different mobile phones and OSs.

Table 2. 2011 Best online data comparisons and reviews [4]

Rank	#1	#2	#3	#4	#5
■■■■ Excellent ■■■□ Very Good ■■□□ Good ■□□□ Fair □□□□ Poor	SugarSync	Dropbox	IBackup	IDrive	Penny-Backup
Overall Rating	■■■■	■■■◧	■■■◧	■■■◧	■■■◧
Backup Features	■■■■	■■■■	■■■■	■■■■	■■■◧
Remote/ Mobile/ Web Access	■■■■	■■■■	■■■□	■■□□	■■◧□
Security	■■■■	■■■■	■■■■	■■■□	■■■■
Ease of Use	■■■■	■■◧□	■■■◧	■■■◧	■■■□
Help/Support	■■■□	■■□□	■■■■	■■■◧	■■■□
Backup Features					
Automatic Backups	✓	✓	✓	✓	✓
Incremental Backups	✓	✓	✓	✓	✓
Selective Backup	✓	✓	✓	✓	✓
File Manager	✓	✓	✓	✓	✓
Supports File Versioning	✓	✓	✓	✓	✓
Archive Folder	✓	✓		✓	✓
Scheduler		✓	✓	✓	✓
Idle Backups			✓	✓	
Remote/Mobile/Web Access					
Internet Accessible	✓	✓	✓	✓	
Folder/File Sharing	✓	✓	✓		
Mobile Phone Access	✓	✓	✓	✓	
Security					
SSL Secure Transfer	✓	✓	✓	✓	✓
Encrypted Storage	✓	✓	✓	✓	✓

As illustrated in table 2, IBackup and IDrive have many similarities, such as automatic backup, scheduling backups, Secure Socket Layer (SSL), secure file transfer, and encrypted storage. Most vendors offer a scalable and on-demand storage space depending on the customer request, offered as a package or specific for a particular client with agreement.

3 Theories and Literature Reviews

3.1 Introduction to Windows Communication Foundation

Over the past decades, procedural and object-oriented (OO) programming paradigms have played a major role in building a variety of distributed systems on top of their pitfalls and limitations. A distributed system can be defined as a collection of autonomous computers that are connected across a network and distribution middleware. Since these computers are integrated and share the same resource and collaborate to accomplish some tasks, they are considered as a single system. [5; 1-43.]

The introduction of service-oriented architecture (SOA) shifted the entire OO concept. The SOA provided diversified support for systems that are running on various platforms with different technologies. SOA is used to build integrated software applications based on a set of ‘services’. A service is an autonomous system which is used to implement a set of published and defined business functionality for clients in various applications. These services can interact with well-defined messaging that can be developed in different types of programming languages and use different kinds of hosts. SOA enhances loose coupling between software components for reusability and boosts interoperability and flexibility between heterogeneous applications. [6.]

In building a service-oriented model based distributed system, service orientation contemplates four tenets during service design. These four tenets are the following:

Tenet 1: Boundaries are explicit.
Tenet 2: Services are autonomous.
Tenet 3: Services share schema and contracts, not classes.
Tenet 4: Service compatibility is based on policy. [7.]

In 2006, SOA was released for the first time as part of the .NET Framework 3.0. This key concept behind WCF primarily is carried as a set of classes on top of the .NET Framework’s Common Language Runtime (CLR) [5]. WCF is used to build service-oriented applications for small or large business processes. WCF allows clients to access services from all kinds of platforms. Wherever it runs, clients and services can interact via Simple Object Access Protocol (SOAP) or/and a WCF-specific binary protocol, and in other ways. [7, 8.]

In addition to this, WCF is also used as an interface for a distibuted model to achieve independent communication between a client and a server. WCF-based clients and services do not require a defined host to run in any Windows process. Clients can instantiate or activate data exchange through messages with the listening server. [6.]

The three main design goals of WCF are the following:

I. Unified programming model: WCF unifies today's distributed technology stacks with composable functionality.
II. Interoperatability across platforms: non-Microsoft platforms and existing Microsoft investments can be integrated and operated via the WCF program.
III. Service oriented development: WCF uses a service-oriented programming model and also supports the four tenets of service-orientation.

Three Components of WCF

The endpoints of the service are the core parts of the WCF communications, which provides clients access to the service functionality. An endpoint consists of an address, a binding, and a contract as shown in figure 1. A client application contains an endpoint with the three core components to use the WCF client to communicate with the service. Services can have multiple endpoints composed of the three components.

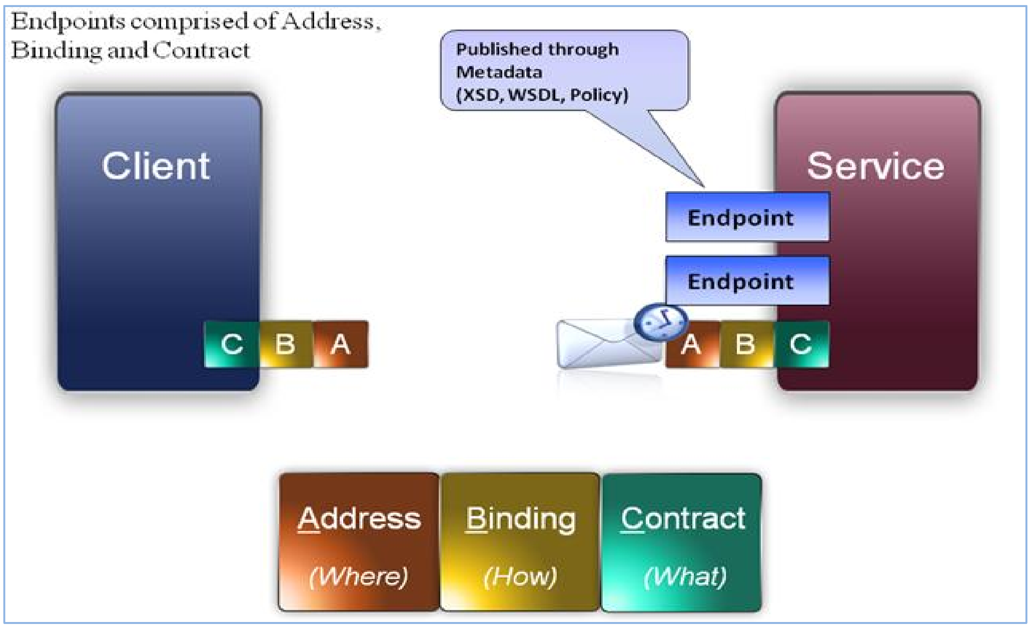

Endpoints comprised of Address, Binding and Contract
Published through Metadata (XSD, WSDL, Policy)
Client
C B A
Service
Endpoint
Endpoint
A B C
Address (Where)
Binding (How)
Contract (What)

Figure 1. Components of endpoint address bind and contract [9]

As illustrated in figure 1, these three endpoint components are address, bind, and contract. They are explained briefly below.

Addresses: The address uniquely specifies the endpoint and tells potential clients where the service can be found or where to send the messages. It is represented in the WCF object model by the endpoint Address class. An Endpoint Address class contains a Uri property, which represents the address of the service, and an Identity property, which represents the security identity of the service and a collection of optional message headers.

Bindings: The binding specifies how to communicate with the endpoint or how to send a client a message. This includes the type of transport protocol to use (such as the TCP or HTTP protocol), the type of message encoding to use (for example, binary or text encoding) and the necessary security requirements (such as the SSL or SOAP message security).

Contracts: The contract describes what functionality the endpoint exposes to the client. A contract specifies operations that can be called by a client, the message form, data required to call the operation or the type of input parameter and the type of processing or the response message that the client can expect. [9.]

3.2 WCF on Linux

Ever since open source (community-built) implementation started delivering more extensible, highly reliable, less-costly and easily integratable software, developers came up with a captivating WCF component known as Mono. Mono is a software platform which can be used to create cross platform applications using Microsoft .Net Framework based on ECMA standards for C# and CLR. It supports Linux distribution, UNIX, Mac OS, Solaris, and other standards. Even though the Mono project is at the early release stage (the latest release is Mono 2.10.5), it contains most of the .NET resource replacements and libraries. [10.]

Mono is distributed in three logical components: the Mono runtime and tools, the Microsoft .NET compatibility API assemblies to provide better functionality, and the Mono

Application Programming Interface (API) assemblies and additional elements to the core Mono. Figure 2 shows the modules of Mono containing a different set of class libraries that constitute the .NET class library implementation, such as ADO.NET, Net, eXtensible Markup Language (XML), collections, and threading. In addition, it also contains GNU Network Object Model Environment (GNOME) and Unix Libraries, such as a GNOME toolkit called Gtk#, that provide a set of C# bindings and integration used to develop GNOME-based native applications for other platforms besides Windows. [11.]

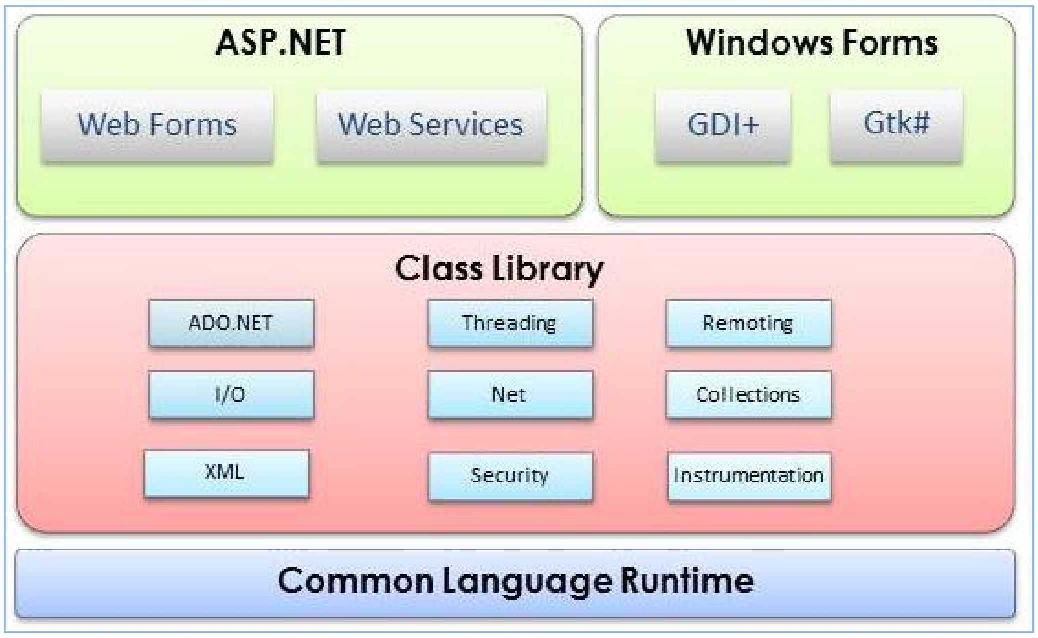


Figure 2. High-level Mono components [12.]

The Mono system consists of Mono C# compiler (MCS), runtime, assemblies (code libraries), and documentation. MCS is the base component of the Mono development environment. Even though the compiler is still in the early development stage, it is compatible with both Java and Visual Basic. The Mono runtime engine provides a Just-in-Time (JIT) compiler, an Ahead-of-Time compiler (AOT), a library loader, a garbage collector, a threading system, remoting, and other interoperability functionalities. [13.]

In addition, the Mono runtime can be used as a stand-alone process, or it can be embedded into applications. Mono also provides bundles that are used to merge (statistically linking) multiple applications, used libraries, and the Mono runtime into a single executable image. Currently, Mono uses the Boehm conservative garbage collector as its garbage collection engine. [13.]

3.3 Client-Server Communication Methodology

Principally there are two types of network paradigms in constructing distributed application architecture to communicate between two edges of a network; these are a client-server model and a peer-to-peer (P2P) model. Both structures use a network system (TCP/IP protocol) to create a network stream and data transaction across the network. The client-server model is a computing architecture based on an asymmetric relationship in which program logics are distributed and shared between a client system and a centralized server system. In the P2P model, clients can initiate unidirectional or bidirectional communication with the other client (a peer) on-demand by sending a dynamic message to the other peers using multicasting or by broadcasting a request.

Unlike the P2P model, in the client-server model, the server determines which users can access resources over the network using authentication keys and a password. For such a model, the server provides all the data to be shared by one or multiple clients from a central storage device. The server also handles and manages data transfer and network security for multiple clients with concurrent requests using distinct IPs. For a client/server scheme, a client has its own customized user interface to initiate and access resources and to execute operations on a remote server. During client requests for services from the listening server, a handshake and authentication are performed and prompted for further operation or, if the connection fails, the clients are disconnected after which they are eligible to send a new request.

3.4 Sockets in .NET

Sockets are low-level network programming features which are used to create an endpoint bidirectional communication between client and server programs using a standard network protocol running on the same network. Sockets are used as a transport mechanism for creating a high-performance communication link between two endpoints (client and server). Sockets are represented by integers commonly known as socket descriptors. Sockets use a transport protocol for exchanging information from one port to another by either a connection-oriented or a connectionless method of communication. [14.]

Depending on the protocol type, there are two common types of communication protocols used for creating sockets in C# programming, TCP based and User Datagram Protocol (UDP) based sockets. The basic difference between these two sockets depends on their implementation technique. UDP sockets are used to transfer connectionless messages for broadcasting and multicasting communication. Moreover, in UDP communication data exchange, there is no active, real time connection between the source program and the destination porgram. On the other hand, the TCP sockets guarantee that the message sent or received on the socket is delivered in an accurate and reliable way by using an error-detection and error-connection mechanism. [14, 15.]

For a client-server model, socket-based methods are applied to initiate a file transfer operation across the network. The .NET framework provides a higher-level abstraction of creating a managed implementation of the Windows Sockets (Winsock) interface using the socket helper classes. These are *TcpClient* class, *TcpListener* class and UdpClient class. However, these classes are missing some of the advanced features of a lower-level socket class provided by the *System.Net.Sockets* namespace. [16.]

Accordingly, in the client-server model, the client and server can create and instantiate a new socket object of the type *Socket* class on both the client and server side. This object can be constructed having three parameters or characteristics defining the socket application and in what approach it can interoperate with another socket application. A socket object is initialized using *AddressFamily*, *SocketType* and *ProtocolType* as as parameters.

Once the *Socket* object is initialized, the three parameters are completed with their respective properties. *AddressFamily*, which designates the addressing scheme used by the socket, can have different types of values. *InterNetwork*, the common *AddressFamily* type, is used for IPV4 addresses. Depending on the type of communication on the created socket object, *SocketType* can have different values of the *SocketType* object. The *SocketType* objects such as *Stream, Dgram, Raw,* and *Rdm* are commonly used. A *Stream* object is applied for connection-oriented byte streams without duplicating data and preserving boundaries on the socket. The third parameter for the *Socket* object, *ProtocolType,* specifies the type of protocol used depending on the *SocketType* [16, 17.]

Sockets can be classified into blocking and non-blocking mode depending on the type of operation performed on that socket. As regards blocking (synchronous) sockets, programs are "blocked" awaiting the request for data until the operation on the socket is fulfilled, but for non-blocking (asynchronous) sockets, the application is allowed to respond to events (asynchronously) upon the completion and execution of the process. This operation is done by using one or two methods for polling an attempt to read and write data and to get notification to recognize error conditions and a successful operation. [18.]

As described in table 3, some of the methods used by the socket can be used only for the client socket or the server socket but in some cases for both. For example, an *Accept* method which can only be called from the server socket object is used upon a new client request to create a new socket. The *Socket.Connect (IPAddress, Int32)* method contains the IP address of the remote host and the port number assigned by the server with an integer value. It is used to establish a synchronous network connection between local endpoint and the specified remote endpoint.

Table 3. Standard socket methods [16]

Method Name	Description
Accept	Creates a new Socket for a newly created connection.
Bind	Associates a Socket with a local endpoint.
Connect (IPAddress, Int32)	Establishes a connection to a remote host. An IP address and a port number specify the host.
Listen	Places a Socket in a listening state.
Shutdown	Disables sends and receives on a Socket.
Close	Closes the Socket connection and releases all associated resources.

Some other methods, such as *AcceptAsync*, *BeginAccept(AsyncCallback,* Object), *BeginDisconnect*, *EndAccept(Byte(), Int32, IAsyncResult),* and *EndReceive (IAsyncResult, SocketError)* can be used to operate on an asynchronous socket object.

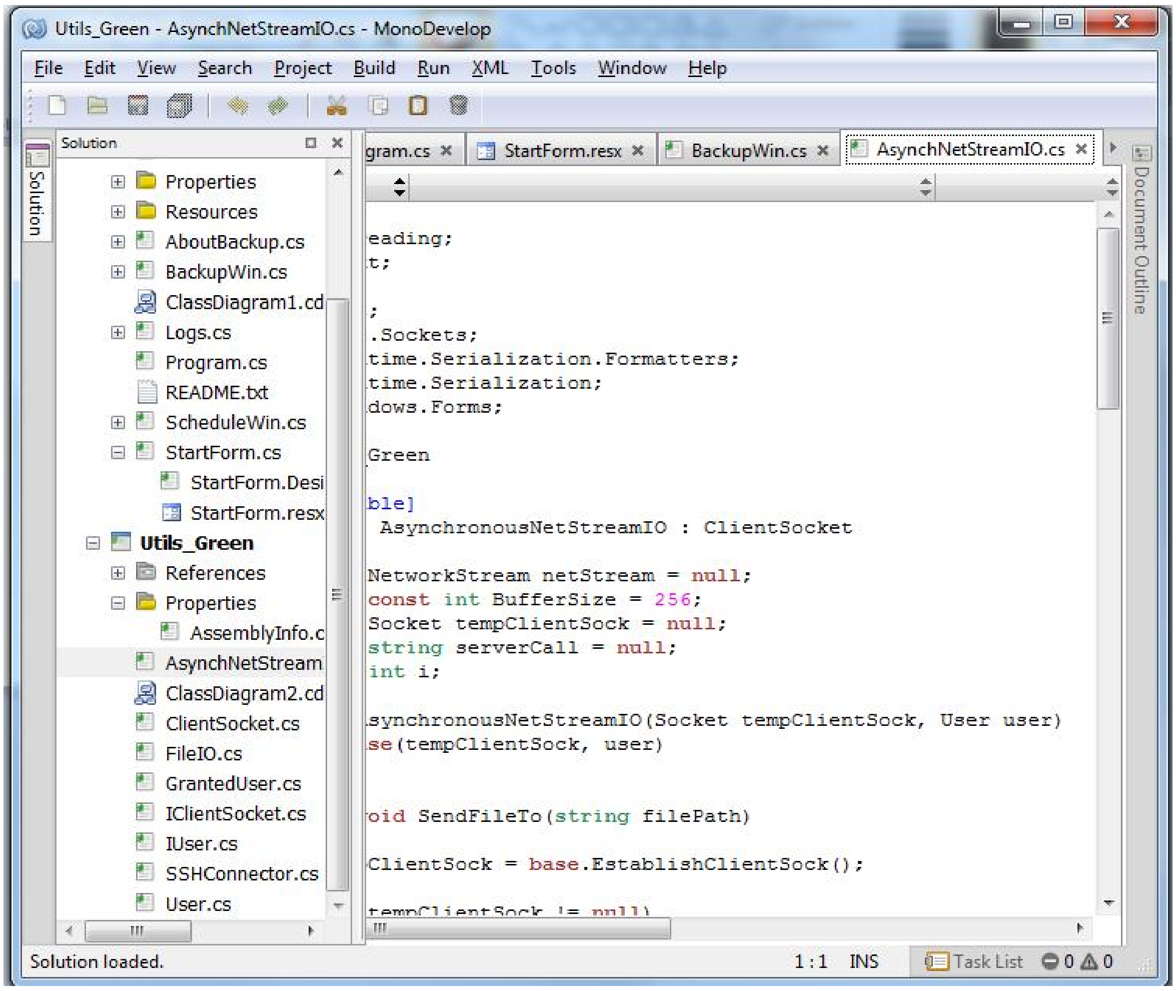
4 Software Design Analysis

4.1 Developing Frameworks and Tools

Online data backup software is intended to run on Windows OS as a front-end and on Linux OS as a back-end, so the following frameworks were selected for designing and implementing the application:

I. CentOS, an enterprise-class Linux distribution providing the backbone for the data backup and clients' data storage.
II. Microsoft Visual Studio 2010, a powerful IDE that ensures a quality code throughout the entire application lifecycle, from the design phase till deployment.
III. The Mono Migration Analyzer (MoMA) tool helps to identify issues that might have been encountered when porting the .Net application to Mono. It helps pinpoint platform specific calls (P/Invoke) and areas that are not yet supported by the Mono project.
IV. Microsoft Office tools were also used to write the software requirement and specification.
V. In addition to these, astah* professional, a software design tool for a lightweight Unified Modeling Language (UML) editor was integrated with ERD, DFD, CRUD and mind mapping features.
VI. MonoDevelop, an Integrated Development Environment (IDE), primarily designed for C# and other .NET languages was selected for developing server-side application. This IDE enables to write and implement different desktop and ASP. NET Web applications on Linux, Windows and Mac OSX. [11]

MonoDevelop is an open source Integrated Development Environment (IDE) for Linux platform users, primarily designed for C# and other .NET languages, but the latest version also supports multiple languages, such as Java, Python, Vala, C, and C++. MonoDevelop enables developers to quickly write desktop and ASP.NET Web applications on Linux, Windows and Mac OSX. MonoDevelop makes it easy for developers to port .NET applications created with Visual Studio to Linux and to maintain a single code base for all platforms. [11.] Figure 3 shows a running Mono project IDE installed on the Linux system.



```
eading;
t;

;
.Sockets;
time.Serialization.Formatters;
time.Serialization;
dows.Forms;

Green

ble]
 AsynchronousNetStreamIO : ClientSocket

NetworkStream netStream = null;
const int BufferSize = 256;
Socket tempClientSock = null;
string serverCall = null;
int i;

synchronousNetStreamIO(Socket tempClientSock, User user)
se(tempClientSock, user)


oid SendFileTo(string filePath)

ClientSock = base.EstablishClientSock();

tempClientSock != null)
```

Figure 3. Mono develop IDE

Feature Highlights

MonoDevelop supports the GUI development with Stetic (MonoDevelop's integrated GTK# visual designer) with abundant functionalities and features compared to the Microsoft Visual Studio 2010.

- Multi-platform: Supports Linux, Windows and Mac OS-X.
- Advanced text editing: Code completion support for C#, code templates, and code folding.
- Configurable workbench: Fully customizable window layouts, user defined key bindings, external tools.
- Integrated debugger: For debugging Mono and native applications
- GTK# visual designer: Easily build GTK# applications

- ASP.NET: Web projects created with full code completion support and tests on XSP, and the Mono web server.
- Other tools: Source control, makefile integration, unit testing, packaging, deployment, and localization. [10.]

4.2 UML Diagrams

The use case diagram shown in figure 4 describes the possible actors and their respective use cases. A new client can download the executable file from the host web site and make the installation. The second actor, the registered client can, for example, login, request for lost password, manage a backup set for a single backup, explore the workstation or backed up files using the application, restore backed up files, send error reports, disconnect from the server, or synchronize backup files on the host.

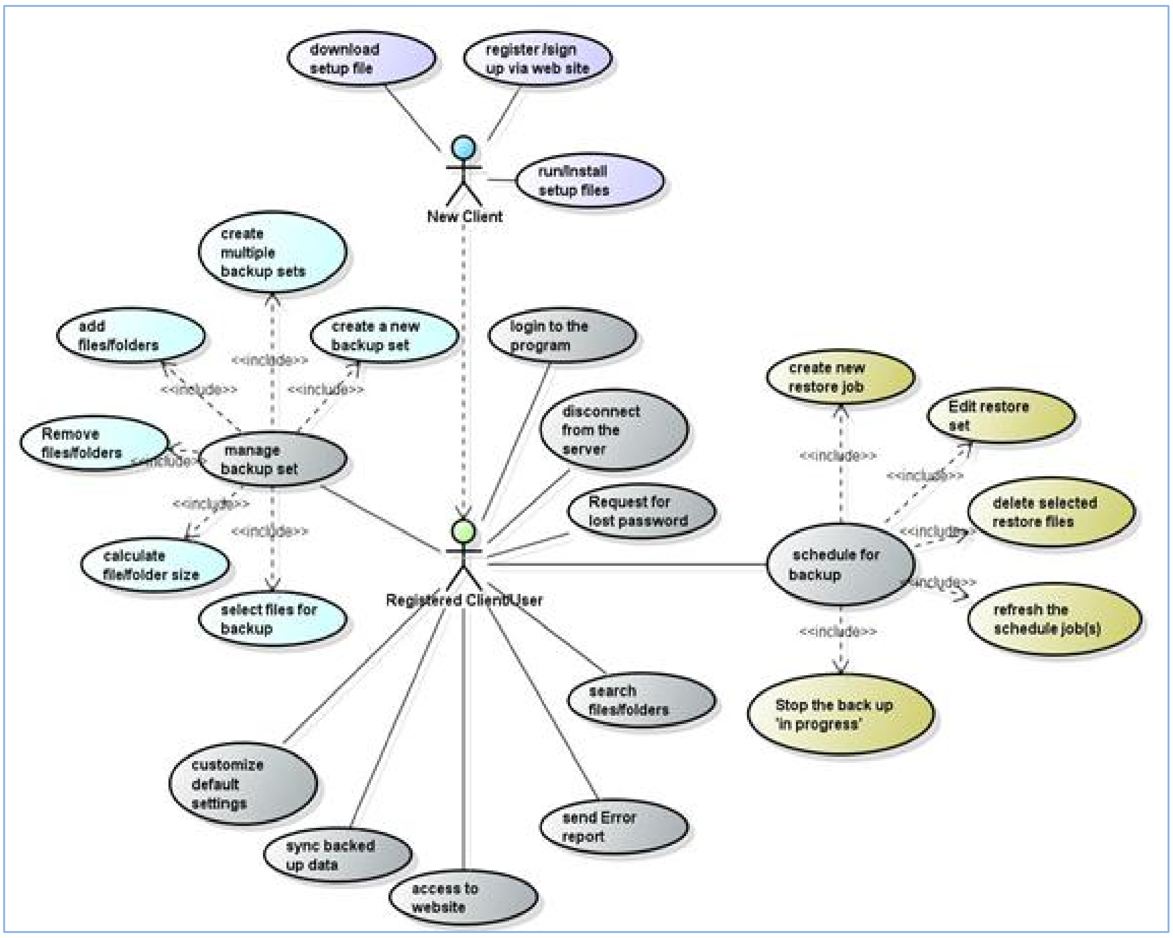


Figure 4. Use-case diagram

Managing backups includes filtering and adding files to the backup set, removing files from the backup set, calculating file sizes, and making schedule backups. Scheduling a back-up includes creating new restore or backup points, editing and deleting scheduled tasks, and stopping pending tasks. The registered client can also create a log file to view the status of the scheduled task, on-demand backup or restore processes.

The sequence diagram in figure 5 illustrates a series of sequential interactions between the client and the backup system. Once a client has registered to the Linux system, the background processes will be automatically handled by the application running at the back-end. Nonetheless, a client can interfere any process being executed on the application layer.

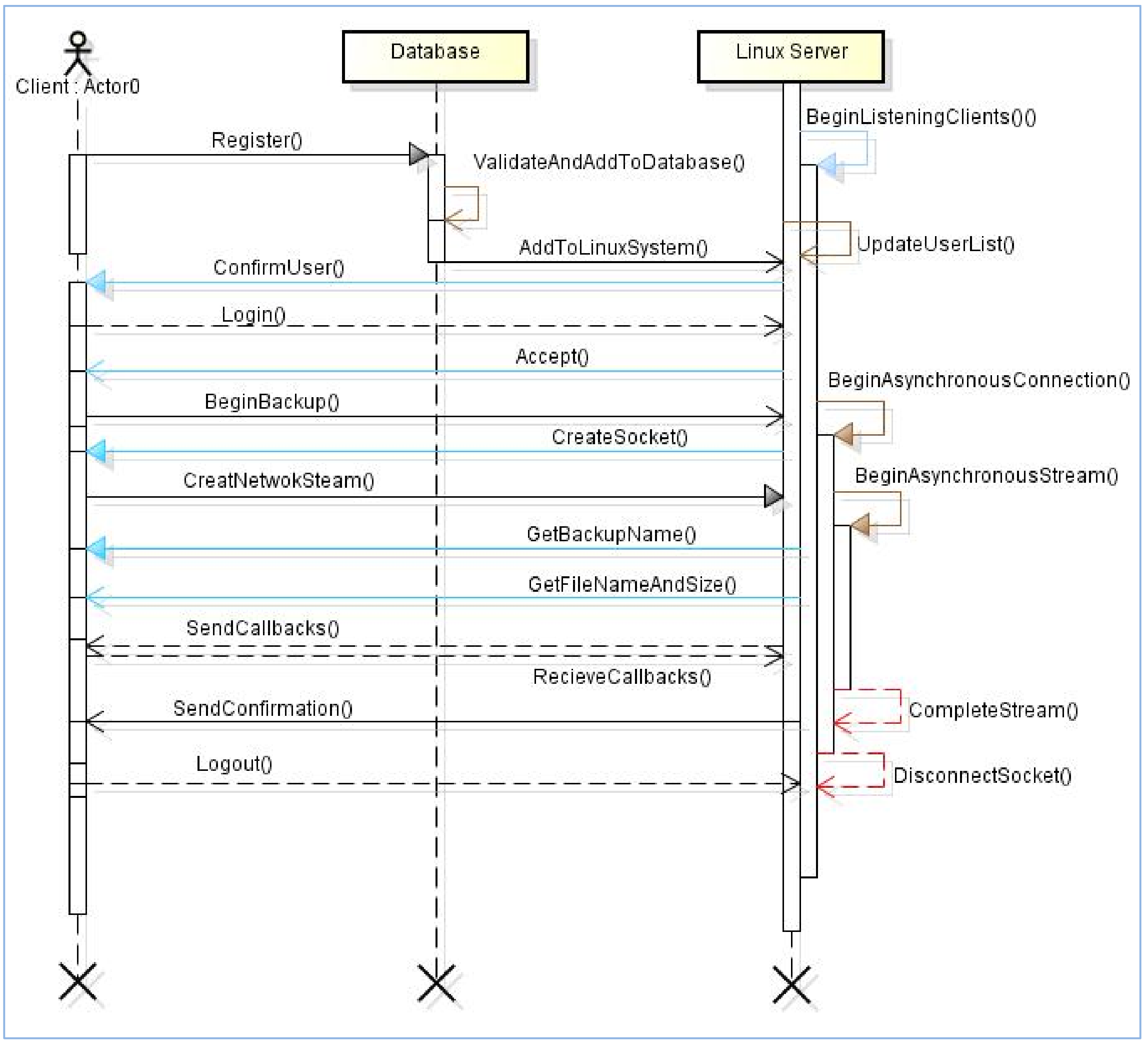


Figure 5. Sequence diagram

Figure 6 illustarates an asychnronous socket activity diagram. The server, which will be running and waiting for a client request at any time, can accept or deny the incoming connections depending on the authentication information sent from the client's program. Thus, this connection creates a new socket object asychnronously with a specific ID to stream files and to hold received bytes from the network stream.

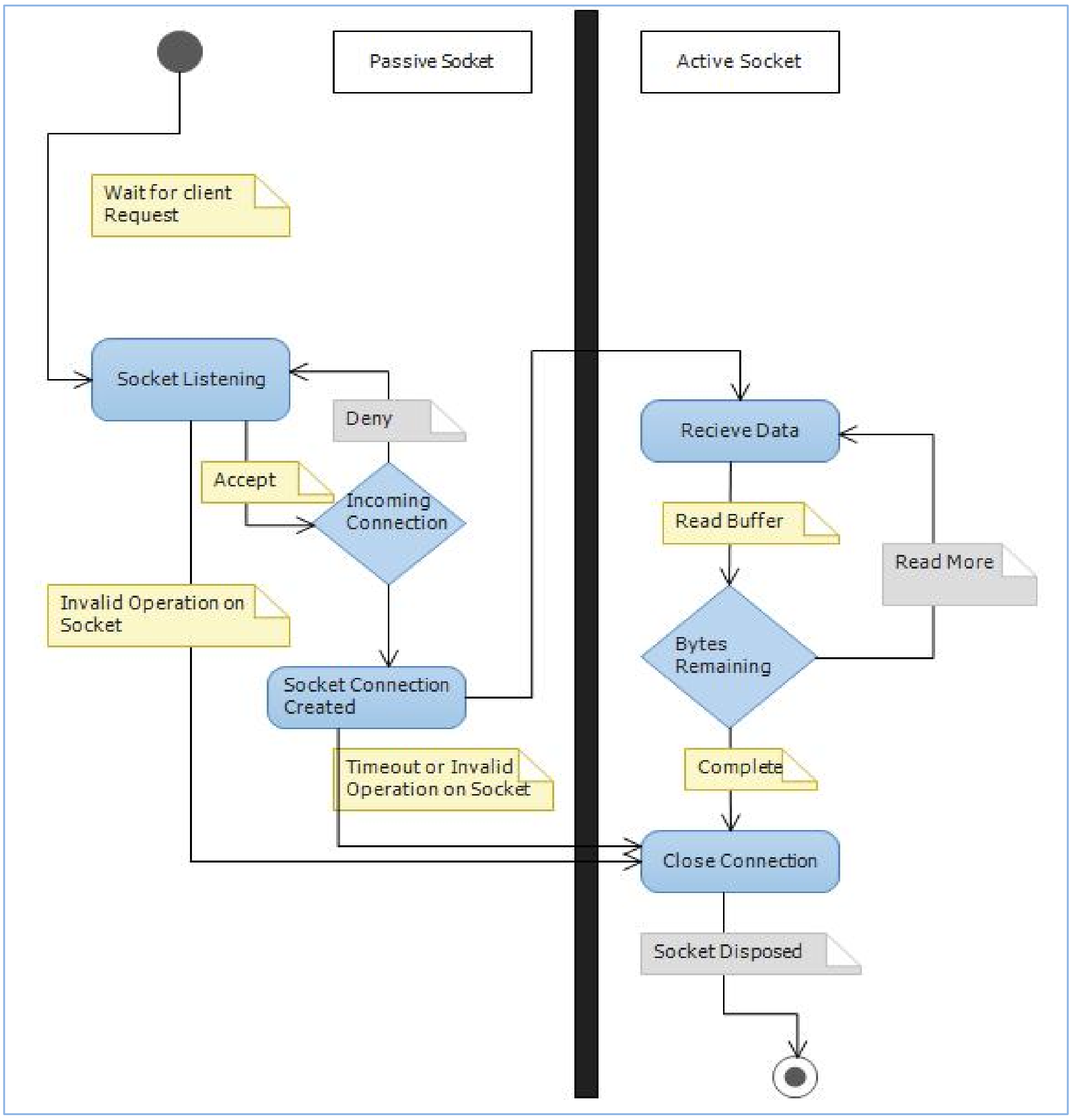


Figure 6. Asynchronous socket activity diagram

An active working socket object is used until the connection timeout ends or is reset by the client's program using the passive socket. After the file stream is completed and all bytes are transferred to the endpoint, the socket is disposed and all the resources that are released will be reused again.

The class diagram in figure 7 describes potential utility classes designed for this application. The first line on top of each class refers to the name of the class. The second segment on the stack represents the attributes of the class basically containing the fields which shall be implemented on another class that inherits them. The methods, which are listed on the third segment of the stack, indicate the class operation (methods) that can be used within or outside the class.

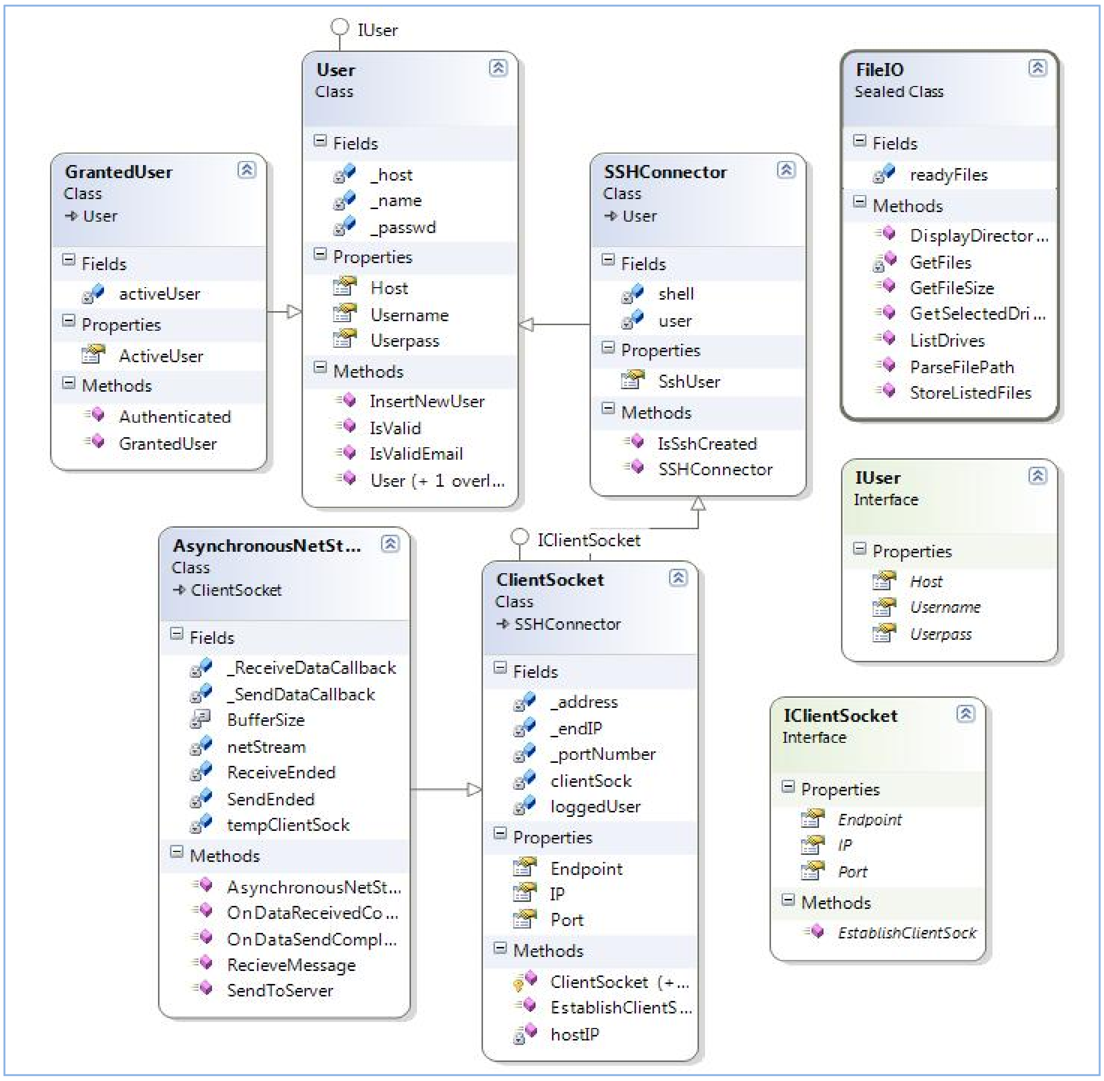


Figure 7. Client Utility class diagram

One class can have an association or inheritance property to access the fields and methods of parent classes to implement a specific task. The *IUser* and *IClientSocket* are interfaces that are implemented on the *User* and *ClientSocket* class respectively. Other classes for this application are listed in appendices 3, 4, and 5.

4.3 Customizing the Software

Online backup software is used to store copies of files and documents of a client on a remote server and to organize data accordingly to be re-used during restoration. The remote server, which is sited in a different location, uses secure file transfer and encryption to minimize security threats. Private clients with a single computer or multiple workstations residing in different locations under the same business group can use the application. The software is mainly intended to collect, compress, and encrypt clients' data and transfer their files to the remote backup service provider's server using an Internet connection. The application can also be used to initiate retrieving backed up files and to manage scheduled backing up on the workstations.

In figure 8, the essential client states are illustrated from the login/register phase until the final step. None of the clients are allowed to create a socket connection to stream files without having the personal data in the host server database with a Secure Shell (SSH) password to access the host. During the file transfer from the client's workstation to the host server, a socket connection is created holding the username sent by the client.

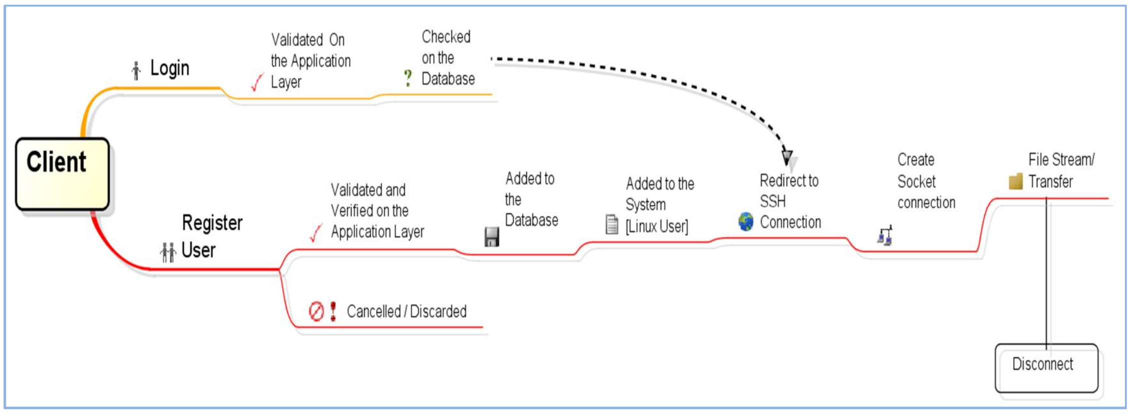


Figure 8. Client-server connection summary

A client trying to operate on a non-created socket will receive an error message sent by the server, and the client will be automatically disconnected. After each backup task, the client will receive a message which confirms how many bytes of data are saved on the server successfully, and the socket resource will be released for future use with connection request.

5 Software Development Approach

5.1 Online Backup Attributes

Graphical User Interface (GUI)

Unlike command line-based backup applications or traditional text-based backup systems on a Linux machine, online backup application is designed to have an interactive and intuitive GUI, which shows the different states of the process and creates a humble and easy usage of the application. The GUI is also be able to display multiple steps and allows complex tasks to be done simultaneously and without difficulty. Starting from the initial installing phase of the software, configuring settings and optional dialogues, users are supplied with an easy access to perform the necessary functionality. Some of these tasks include configuring and managing accounts, choosing optional tools, searching files, creating desktop short-cuts, setting backups, restoring sets, and creating scheduled jobs.

Scheduled and On-Demand Backup

The clients are provided with different options to schedule time and the date for the execution of the backup or restoration by means of GUI. The data to be backed up is encrypted and transferred securely to the remote server at a specific time. A scheduler is used to run backup jobs automatically as a background task to save time during off-peak hours. Clients can backup data once, daily, weekly or monthly depending on their choice and Internet connection speed. The Windows task scheduler or the Linux crontab can be integrated with this application to perform scheduled tasks at regular intervals.

Security

The security of online backup is treated as high priority, and it is impossible for valuable information to fall into the hands of unauthorized personnel. Therefore, data is transferred in a compressed and encrypted state via the SSL connection to highly secured and replicated data servers. Every online backup account owner can limit access and restore to a predefined list of accounts.

An encryption key, which can be generated by the client, is only known by the client and will never be sent via the Internet. If the encryption key is lost, the backup files cannot be restored. Enhanced security with 128-bit SSL encryption on transfers and optional 256-bit proprietary encryption on storage with a user-defined encryption key that is not stored anywhere on the remote servers are used. To establish the secure online Internet connection, this application uses an SSL connection, which is similar to the one that banks use for online banking.

Filtering

There are a couple of ways to select files for backup, such as using a directory tree, or for advanced users, by creating rules. All files are de-selected in the same way that they are added. Before selecting files to backup, it is important to organize the data for backup. Filtering lets the user determine which types of files should be included in or excluded from the backup set since all files on the client computer are sorted accordingly. Filtering enables clients to select files of a specific type to be visible on the clients' GUI to be included or excluded from the backup set. Some examples of file extensions are given below:

- OS files such as ones that end in DLL, SYS, CPL, or VXD.
- Application or program files such as ones that end in EXE, INI, HLP, or DAT.
- Scan disk error files such as files ending in LOG.

Filtering can be done in two forms, by the types of files according to their extension or by the size of the files. Filtering can also be done using the search tool to locate a particular file easily.

5.2 Data Encryption and Security

Storing critical information under the file of an executable-hosted application after the deployment and installation can expose hidden files since any user can decompress and unpack a self-extracting *.exe* file or an application installer easily and use credential information for hacking the database server or file server. To get rid of such data exposure, most of the crucial data sent over the network should be encrypted or hidden from the user.

One of the common protocols used to deal with secure and encrypted network communication is known as SSH. SSH can be used for secure logging into a remote host, file streaming and transferring, running remote commands without manual authentication and encrypted tunneling. SSH is the replacement of non-encrypted protocols such as File Transfer Protocol (FTP), Telnet, and Remote Shell (RSH) programs.

However, the .NET framework does not support native SSH implementation; therefore, a third-party library called SharpSSH is used to carry out secure communication between the server and the client. SharpSSH is an API for creating a messaging channel for running a shell on a remote SSH server which is implemented with pure .NET and is capable of being integrated with any .NET application. The library also contains a C# port of the JSch project from JCraft Inc. and is released under a Berkeley Software Distribution (BSD) style license. In addition, SharpSSH allows exchanging data and transfer files over SSH channeling with additional wrapper classes. [19.]

Some of the features provided by SharpSSH are listed below:

- Secure File Transfer Protocol (SFTP) refers to secure FTP using SSH by encrypting data to prevent sensitive data while transferring over the network.
- Secure Copy (SCP) is secure file copying technique between two or more hosts on a network using an SSH password or passphrases with authentication.
- Cipher is an algorithm used for session encryption during SSH channeling. Some of the algorithms that are supported by SharpSSH are des-cbc, aes128-cbc.
- Message Authentication Code (MAC) refers to constructing cryptographic hash function in combination with a secret key to use Hash-based MAC.
- Generating key pairs refers to generating private and public keys on UNIX based systems to provide greater security when logging into a server using SSH. For instance, Rivest, Shamir and Adleman (RSA) and Digital Signature Algorithm (DSA) keys are commonly used.
- Passphrase means generating a random key for the symmetric cipher for the password. SharpSSH also provides a method to change private keys.
- Port and stream forwarding refers to the process of redirecting computer signals to use the right kind of network data on the right port. [19.]

Moreover, since the application is installable on a client's computer, sensitive configuration setting files must be hidden from the user. Configuration files such as database connection, which contains confidential information, can be customized by implementing an XML file. It is important to detach the file from the application to be only executed and used at runtime. The source code in listing 1 shows the name of the connection string, the data source, the user ID, the name of the database used, and the type of the database source.

```
<?xml version='1.0' encoding='utf-8'?>
<configuration>
  <configSections></configSections>
  <connectionStrings>
    <add name="Backup_Users" connectionString="Data Source =   gsbackup.greenspot.fi;
                                       user id =  gsusers;
                                       database = Greenspot;
                                       password = *********;"
         providerName = "System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

Listing 1. Database connection string (App.config)

To carry out this in practice, an application configuration file named after the application name (Uifs_Green) is added to the application with a .config file extention. This assembly file will only be located and loaded during CLR. Customizing an application setting is also important to easily maintain or set up a new configuration for future updates. The connection string instance containing the database connection string object, shown in listing 2, is used when there is a need to fetch data from the '*Greenspot'* database and insert data into the database.

```
string connectionString =
ConfigurationManager.ConnectionStrings ["Backup_Users"].ConnectionString;
```

Listing 2. Database connection string object

The *ConfigurationSettings.AppSettings* class in the *System.Configuration* namespaces natively provides a simple mechanism to hold essential key-values and utilize them in the .config file of an associate application. The key-values are simply stored in string format and can easily be retrieved by using the *ConfigurationSettgins.AppSettings* object as listed in listing 2.

5.3 Server Configuration

All users' data and back-end operations are executed and monitored on the server side. Thus, a complete functional server is set up with operative configuration files and scripts running in the background to handle multiple clients. Configuring the server is done in three phases.

Phase 1. Installing and configuring Apache and MySQL servers

To work with the database and to host a web-based application, an Apache server is a prerequisite. Listing 3 indicates the final steps to checkup the Apache and MySQL server installations using the repositories and software packages provided by CentOS.

```
~# httpd -v
Server version: Apache/2.2.3
~#mysql -V
mysql Ver 14.12 Distrib 5.0.77, for redhat-linux-gnu (x86_64) using readline 5.1
```

Listing 3. Apache and MySQL installation

Phase 2. Mono and other packages

As explained in chapter 3, section 3.3, to compile and run the C# application on a Linux machine, a Mono package is required to be installed on the server. Thus, after downloading the source code (mono-2.10.tar.bz2) from *http://ftp.novell.com/pub/mono/download-stable/RHEL_5/repodata/* it is stored under the /usr/src/ directory and unpacked under the mono-2.10.tar.bz2 directory using a command called *tar*. Then the configuration steps in listing 4 are done for configuring and installing the *mono* compiler on the Linux system. After the compiler is successfully installed and configured, the C# console application is complied and run using the mcs and mono commands respectively.

```
~#.cd mono-2.10
~#./configure --prefix =/usr/local
~# ./configure --prefix=/usr/local
~# make
~# make install
```

Listing 4. Configuring and installing mono

Moreover , g++ and gcc, two necessary Linux packages to compile and run all GCC compilers (namely C++, Fortran 77, Objective C and Java) were not installed on the Linux machine. These compilers are used for integrating all the optimizations and features necessary for a high-performance and stable development environment. These packages are also required for the GNU C++ compiler used by Mono. In addition to these compliers, the Linux machine complained about the parser to get the mcs compiler work effectively. Thus, Bison, a general-purpose parser to generate C program grammar, was installed successfully [20.] In listing 5, the main steps are shown on how to install these compilers and the Bison parser to the Linux machine.

```
gcc:  ~# yum install gcc
g++:  ~# yum install gcc-c++
bison: ~# yum install bison
```

Listing 5. Complier and parser installation

Phase 3. Automating tasks

When the clients/users attempt to login or register via the login/register form, they are automatically validated on the client's application itself and the data will be sent to the server database. This database contains essential user information for verification purposes to prompt the user to the main backup window. Subsequently, the user is added to the Linux user list with their respective username and password using bash shell scripting, by creating a temporary file named as queryOutput.log as shown in listing 6. Then this temporary file is stored under the >/tmp/ system file.

```
get_user()
{
# using the password, login to mysql without prompt and query users and save usage
  mysql -u root –p****** Greenspot <<EOFMYSQL  >/tmp/queryOutput.log
  SELECT userName, passwd, emailAdd FROM GsBackupUsers;
EOFMYSQL
}
```

Listing 6. Users information fetch from database (doauto.sh)

The client tasks, which are used to login, register or transfer data to the server, are not done manually for this application. Thus, the server application running continuously at the background is automated using a cron daemon provided by the Linux machine. This utility is set up on the Linux server to automate selected server scripts and allow tasks to run at regular intervals. To implement cron entities, a bash shell script is used to generate the executable file and added to the cron tab list using the command *crontab -e* by the root user.

In doing so, the clients/users are automatically checked on the Linux SSH users' list to permit SSH access for the client application. As illustrated in listing 7, the newly fetched users from the database, which are saved in the queryOutput.log log file are iterated and matched with the user list extracted from the /etc/passwd file. Then new users are added to the system with a respective home directory and an old user's password will be updated. Finally, the temporary file that stores the user's information will be automatically removed from the system file by the root user.

```
# Add new user to Linux box which are retrieved from the database-> automatically
main() {
# get existing ssh users inside the linux box
export SSH_USERS=$(cat /etc/passwd | grep 501 | cut -d: -f1 | sort)
# users who are newly registered users->add to db and query is saved under /tmp/ file
export NEW_USER=$(cat /tmp/queryOutput.log | cut  -f1 | more +2  | sort)
export NEW_USER_LIST="/tmp/queryOutput.log"
export HOME_DIR="/home/"
                more +2 ${NEW_USER_LIST} | #Skip the column headers
                        while
                        read userName passwd emailAdd
                        do
                                for user in ${SSH_USERS[@]};
                                do
                                        if [[ $user != $userName ]];
                                 then
 # add user to linux box with respective username and email address
  # home directory will be created under /home/username with default group gsclient
                              /usr/sbin/useradd -g gsclient -s /bin/bash -m -d ${HOME_DIR}
${userName} ${userName};
                             echo ${passwd} | passwd --stdin ${userName};
 # store logs of registered users
                                 echo "$userName registered successfully at $(date)" >
/root/gsbackup/logs/register.log;
                                else
                                   echo $userName "already exists!";
                                        fi;
                                done;
                                #remove username list from the temp file
                                rm -rf /tmp/queryOutput.log;
                        done;
                        exit 0;
}
```

Listing 7. Bash shell script (automatic user adder to Linux system: doauto.sh)

In typical service oriented architecture new client/user information is stored on the remote server database, so that clients are prompted to use services provided by the host. To store these data, users are added during the registration phase and automatically stored in the host database (*Greenspot*). As described in chapter 5, section 5.2, the client's application uses the connection string object to insert or fetch credentials from the database using configuration settings.

6 Backup Technique

6.1 Clients' Interface and Usability

As a logged-in user, the startup form (window) will be redirected to the main backup window as shown in figure 9. This window contains the basic and advanced features of the application providing easy and straightforward access to the functionalities and usability of the application. It has standard window formats including the menu bar, tab tools, user information, drive selector, status bar, and the task bar.

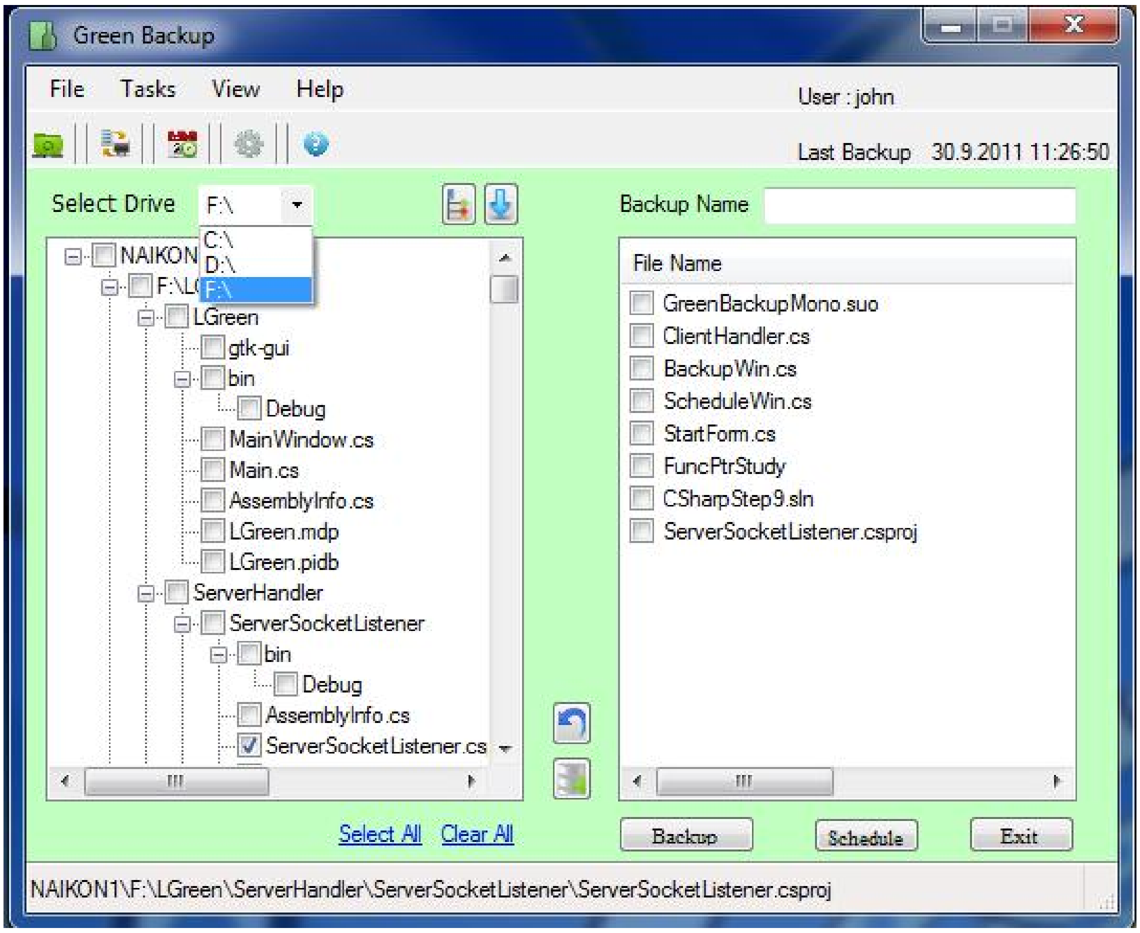


Figure 9. Main backup window for clients

The backup window shown in figure 9 provides a pleasant appearance to the user for searching files through folders by expanding and selecting particular files for backup. The undo button is used to deselect unwanted files from the backup list box. Each backup set will be saved to the server with the backup name and date.

6.2 Asynchronous Sockets

Hence, data transferring depends significantly on the use of the network, and the server and clients are supposed interact using events and triggering delegates. The server stays in a listening state for client requests as it is consistently waiting for multiple clients' calls. To manage these clients, event-driven programming is applied to monitor multiple client-server communications based on an asynchronous socket programming. This allows the application to continue in a non-blocking mode on a separate thread and none of the clients' sockets is suspended while waiting for the network operation to complete. Asynchronous operation is more efficient and scalable than the fork and threads that create a new process and a new thread respectively every time clients call for a socket connection. Listing 8 illustrates an asynchronous socket which is used to listen and bind the incoming client's address.

```
private void AcceptSocketConnection()
{
  // create socket object
  listenerSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream,    Proto-
colType.Tcp);
   // and bind the socket to the endpoint and wait for any incoming connections
  RemoteEndpoint = new IPEndPoint(IPAddress.Any, Port);
       try
       {
         // bind the remote ip and port
         listenerSocket.Bind(RemoteEndpoint);
         // start listening
         listenerSocket.Listen(backlog);
                   // keep listening
                   while (true)
             {
                    // set the event to nonsignaled state
                    ConnectionDone.Reset();

                  // start an asynchronous socket to listen for any connections
              Console.WriteLine("Server listening and waiting clients...");
              listenerSocket.BeginAccept(_clientConnectCallback, listenerSocket);

               // wait at least for one client request
               ConnectionDone.WaitOne();
             }
         }
}
```

Listing 8. Asynchronous socket Accept method (*AsynchNetStreamIO.cs* Class)

For this application, asynchronous sockets are defined on the client and server end-point to allow communication across the network. Once a socket object is instantiated having the three parameters described in chapter 3, section 3.4, an object of the type *IPEndPoint* class with a specific port number and the IP address of any client is bound to the socket and starts listening to the client/s calls. As shown in listing 8, the bind method of the socket accepts the endpoint as an argument. [21.]

To begin accepting a client request, an asynchronous method named *BeginAccept()* is called. This method contains a delegate method named *AsyncCallback()*, which is used to complete the function (*AsychnCallback()*), and a generic state object that can pass information between the asynchronous methods (*listenerSocket()*) received from the client. Some of the commonly used asynchronous socket methods are listed in table 4.

Table 4. Selected Socket methods [21]

Name	Description
BeginAccept(AsyncCallback, Object)	Begins an asynchronous operation to accept an incoming connection attempt.
EndAccept(IAsyncResult)	Asynchronously accepts an incoming connection attempt and creates a new socket to handle the remote host.
BeginDisconnect	Begins an asynchronous request to disconnect from a re-mote endpoint.
EndDisconnect	Ends a pending asynchronous disconnect request.

During data transfer, using methods that utilize the *AsyncCallback()* delegate to call on the completion method, the network streaming operation is completed to monitor the client. Then the newly created temporary socket object in the method *OnClientSocketConnect()* is called to end the *BeginAccept()* method. In a similar manner, the *EndAccept()* asynchronous call back method is used to end a client's socket connection after the client is successfully connected with the correct socket ID. The *BeginDisconnect()* and *EndDisconnect()* methods work in a similar manner during connection termination and releasing a resource from that specific socket.

6.3 Asynchronous I/O Stream

Data representation is one of the major issues to be handled while transferring data from the Microsoft Windows OS platform to Linux and vice versa. Thus, implementing an interoperable data transfer requires data conversion and serialization to read the file stream from the network, converting back to the original, and to deserialize to the original file stream for the end user. Serialization and deserilazation depends on the direction of the file stream sent over the Internet. To put this into action, a standard .NET Framework network stream mechanism is used to access to the network data. The *System.Net.Sockets.NetworkStream* namespace provides a network stream class to carry out this mechanism. [22, 23; 81-114.]

As mentioned in section 6.2, network streams can also be used in two techniques in a similar way as socket objects: asynchronously or synchronously. The synchronous stream technique is simply creating a new thread that operates upon calling a method and waiting till the operation is completed or failed. On the contrary, an asynchronous stream returns from the method call instantaneously and none of the operations waits or suspends to complete the task until the call back method sends a signal (state object). Table 5 describes the most important members of the asynchronous stream used to read and write in an asynchronous approach.

Table 5. Significant members of an asynchronous stream [22]

Name	Description
BeginRead	Begins an asynchronous read operation.
BeginWrite	Begins an asynchronous write operation.
EndRead	Waits for the pending asynchronous read to complete.
EndWrite	Ends an asynchronous write operation.

After the socket is created and communication is successfully made between the client and the server, the next step is to stream data over the created instance of an asynchronous socket class. To achieve an asynchronous network streaming, a new *NetworkStream* object is constructed holding the specified socket with the stated socket ownership as a parameter. This *NetworkStream* object is used to implement a streaming mechanism for reading and writing data to and from the network by using the members mentioned in table 5. The files which are collected as a list in the list box of the main backup window are iterated and the full path is returned by the *GetFull-FilePath()* method.

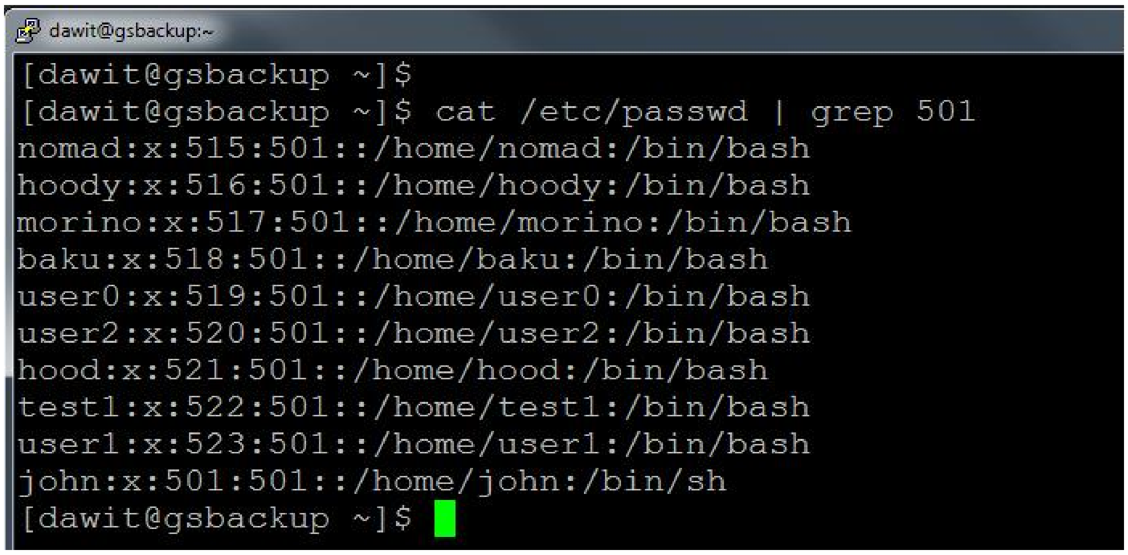
7 Test Cases and Results

7.1 User Interface Testing

The application was installed on the Windows 7 platform for testing. As shown in chapter 6, section 6.1, figure 9, the main backup window contains easy and pleasing menus and tabs for exploring folders and files, check-boxs for selecting and deselecting files, several buttons for making a backup, restoring and creating a scheduled task, correspondingly. The user's basic information is displayed at the top right corner of the window. The status bar is beneficial to indicate the user's current path to the file or/and directory. Users were also provided the ability to switch between the local drives and easily make files available for backup selection. This piece of software has provided users with captivating and forthright features to backup easily.

7.2 Unit Testing

When a new user is registered and added to the Linux system, the retrieved information from the output file, as indicated in chapter 5, section 5.3, listing 7, the user will automatically be added to the group *gsclient*. This user is only allowed to login using an encrypted password and allowed a login shell type (*/bin/bash*). Figure 10 lists users that belong to the *gsclient* group.



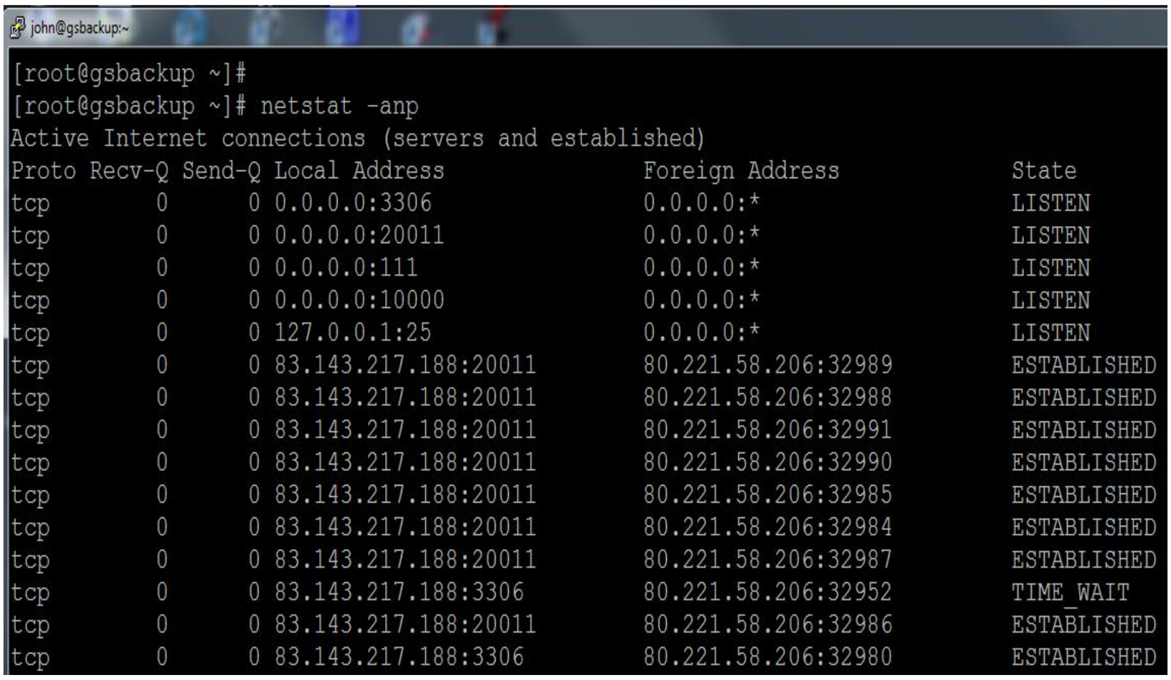
```
[dawit@gsbackup ~]$
[dawit@gsbackup ~]$ cat /etc/passwd | grep 501
nomad:x:515:501::/home/nomad:/bin/bash
hoody:x:516:501::/home/hoody:/bin/bash
morino:x:517:501::/home/morino:/bin/bash
baku:x:518:501::/home/baku:/bin/bash
user0:x:519:501::/home/user0:/bin/bash
user2:x:520:501::/home/user2:/bin/bash
hood:x:521:501::/home/hood:/bin/bash
test1:x:522:501::/home/test1:/bin/bash
user1:x:523:501::/home/user1:/bin/bash
john:x:501:501::/home/john:/bin/sh
[dawit@gsbackup ~]$
```

Figure 10. SSH users from group *gsclient*

The *cat* command is used to list the file under the */etc/passwd* path which contains the list of user details on the Linux server. The option -*grep 501* syntax is added to filter the users who are belonging to the group (*gsclient*) with ID 501. Figure 10 describes a specific user name, an -*x* character indicating an encrypted password, that is stored in the */etc/shadow* file, including the user ID (UID), the user default group ID (GID), the home directory and the users' absolute path of command or login shell respectively.

7.3 Integration and System Testing

For an integrated testing, multiple clients were allowed to login to the system simultaneously and begin file streaming. The result shown in figure 11 describes the list of the socket connections which were successfully made using the intended port (20011) with a distinct socket id from multiple clients. This test was done using the *netstat –anp* Linux command. This command is used to display such things as network connections, interface statistics, routing tables, and masquerade connections. The options used, -*anp,* are used to show both listening and non-listening sockets, numerical addresses and the PID and name of the program to which each socket belongs respectively.



```
[root@gsbackup ~]#
[root@gsbackup ~]# netstat -anp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
tcp        0      0 0.0.0.0:3306                0.0.0.0:*                   LISTEN
tcp        0      0 0.0.0.0:20011               0.0.0.0:*                   LISTEN
tcp        0      0 0.0.0.0:111                 0.0.0.0:*                   LISTEN
tcp        0      0 0.0.0.0:10000               0.0.0.0:*                   LISTEN
tcp        0      0 127.0.0.1:25                0.0.0.0:*                   LISTEN
tcp        0      0 83.143.217.188:20011        80.221.58.206:32989         ESTABLISHED
tcp        0      0 83.143.217.188:20011        80.221.58.206:32988         ESTABLISHED
tcp        0      0 83.143.217.188:20011        80.221.58.206:32991         ESTABLISHED
tcp        0      0 83.143.217.188:20011        80.221.58.206:32990         ESTABLISHED
tcp        0      0 83.143.217.188:20011        80.221.58.206:32985         ESTABLISHED
tcp        0      0 83.143.217.188:20011        80.221.58.206:32984         ESTABLISHED
tcp        0      0 83.143.217.188:20011        80.221.58.206:32987         ESTABLISHED
tcp        0      0 83.143.217.188:3306         80.221.58.206:32952         TIME_WAIT
tcp        0      0 83.143.217.188:20011        80.221.58.206:32986         ESTABLISHED
tcp        0      0 83.143.217.188:3306         80.221.58.206:32980         ESTABLISHED
```

Figure 11. Console output: Active clients connected to the server

8 Discussion

One of the main intentions in carrying out this project was to create fully functional and operational software to the client and to draw a brief conclusion on developing a WCF application on Linux Server. Besides, learning and extending the knowledge of advance C# programming techniques was of additional importance.

While carrying out this project and developing the application presented here, setting up the limitations and constrains in the software specification phase was a challenge; hence, the software has the ability to expand easily and to get out of control. Thus, establishing a decent ground on the existing systems and deciding on the development framework took ample time. In parallel to this, the restriction to deploy the application on a Linux platform was also a determinant factor in obtaining successfully working software. Besides, establishing a conclusive argument in developing a standalone application rather than a web-based application was taken into account.

Although backing up data can be taken as a simple and easy task, many operations are encountered in the process. In developing this application, intensive networking programming and configuring the server were performed in for creating an asynchronous socket connection for clients and handling multiple clients concurrently for the network file stream. During the testing phase, performance and security of the software were achieved with a complete working GUI for the client. However, the heavy-load of the thread on the automatic running script on the server side was considered to be a disadvantage of this application. Nonetheless, the project thrived and managed to achieve the primary purposes of the application.

In summary, even though this backup system was particularly designed and implemented for a Windows platform, by adding missing libraries in Mono, it can be further re-built to support other platforms, such as Linux and MacOS. The application can also be applied to backup files from one drive to another drive or external disk without Internet connection. Bandwidth throttling and extra features and tools can be added to this application before it is released. Once Mono core is supported with the set of add-on libraries and functionalities to provide .NET 3.0 APIs, the application can be designed in different and flexible approaches.

9 Conclusion

The goal of the project was to design and implement a standalone desktop application for an online backup and for the restoring of clients' data to an off-site host provider. The application uses a client-server paradigm to create and implement an interoperable Microsoft Visual C# Windows application and a Linux platform using Mono Compiler between the client and server. In addition to these, it also intended to build a general background for developing a WCF application and a .NET framework on a Linux platform for building SO applications using a unified programming model.

The application was implemented using the Microsoft Visual C# 2010 programming language for GUI design and handling the functionalities and features and a bash shell scripting for automating and handling clients. It was deployed and tested on a Windows 7 platform as the front-end and on CentOS Linux distribution as a back-end. In result, the clients were able to register into the database and login/logout gracefully. Registered users were also added to the Linux system automatically using the automated crontab task on the server. The clients were also able to initiate a backup and customize a backup scheme to fit their needs by selecting particular files and folders for the backup set. Moreover, file streaming and creating an asynchronous socket for each particular client was successfully managed, so that multiple clients residing in a similar location or a different network could access and share files or folders from the host. However, due to time constraints, the project was limited to achieving only the main functionalities: backup, GUI designing, and Mono implementation.

To summarize, this application can be further developed to enhance performance and security and to provide an easy and vast set of tools, such as a real-time backup tool, command line tool, and encryption key generator. It can also be developed to be integrated with full web access and with mobile phones. The application can easily be transformed to a WCF application for any platform as requested by the client, once the Mono libraries are fully supported. Depending on the bandwidth and memory, this application can be optimized to enhanced band width-throttling or dedicating/allocating specific amounts of computer resources for this service and increasing the competence of the software.

References

1 Gogs. Backup basics and different types of backup [online]. Debian Admin; 27 October 2006.
URL: http://www.debianadmin.com/backup-basics-and-different-types-of-backup.html.
Accessed 3 April 2011.

2 Microsoft Corporation. Backing up and restoring data [online]. Microsoft TechNet; 3 November 2005.
URL: http://technet.microsoft.com/en-us/library/bb457113.aspx.
Accessed 4 April 2011.

3 Microsoft Corporation. Description of full, incremental, and differential backups [online]. Microsoft support; 15 November 2006.
URL: http://support.microsoft.com/kb/136621.
Accessed 30 March 2011.

4 TechMediaNetwork. Online data backup review [online]. TopTenReviews; June 2011.
URL: http://online-data-backup-review.toptenreviews.com/.
Accessed 30 June 2011.

5 Chappell D. Introducing Windows Communication Foundation. USA: Microsoft Corporation; September 2005.

6 Evdemon J. Principles of service design: Service patterns and anti-patterns [online]. MSDN Library; August 2005.
URL: http://msdn.microsoft.com/en-us/library/ms954638.aspx.
Accessed 4 April 2011.

7 Mahmoud H. Service-Oriented Architecture (SOA) and web services: The road to Enterprise Application Integration (EAI) [online]. Oracle; April 2005.
URL: http://www.oracle.com/technetwork/articles/javase/soa-142870.- html.
Accessed 1 June 2011.

8 Nishith P. Pro WCF 4 practical Microsoft SOA implementation. New York: Apress Inc.; 2011.

9 Microsoft Corporation. Endpoints: addresses, bindings, and contracts [online]. USA: MSDN Library.
URL: http://msdn.microsoft.com/en-us/library/ms733107.aspx.
Accessed 26 April 2011.

10 Clemens V. and Newtelligence AG. Introduction to building Windows Communication Foundation Services [online]. USA: MSDN Library; September 2005.
URL: http://msdn.microsoft.com/en-us/library/aa480190.aspx.
Accessed 24 April 2011.

11 What is Mono [online].
URL: http://www.mono-project.com/What_is_Mono.
Accessed 20 March 2011.

12 Tapas P. Productivity improvements in Mono 2.4: Components and architecture 2010 [online]. QuinStreet Inc.
URL: http://www.devx.com/opensource/Article/43410/1954.
Accessed 4 May 2011.

13 Hutchinson M. MonoDevelop documentation [online]. MonoDevelop; 6 December 2010.
URL: http://monodevelop.com/Documentation.
Accessed 2 March 2011.

14 Daryn K. Get closer to the wire with high-performance sockets in .NET [online Magazine]. Las Vegas, Nevada; MSDN Library; August 2005.
URL: http://msdn.microsoft.com/en-us/magazine/cc300760.aspx.
Accessed 20 August 2011.

15 Create a connectionless Socket [online]. IBM.
URL:
http://publib.boulder.ibm.com/iseries/v5r2/ic2928/index.htm?info/rzab6/rzab6connectionless.htm.
Accessed 23 June 2011.

16 Houcks. System.Net.Sockets Namespace [online]. MSDN Library; 27 June 2011.
URL:
http://msdn.microsoft.com/en-s/library/system.net.sockets.socket_methods.aspx.
Accessed 6 June 2011.

17 Kim P. Socket class [online]. MSDN Library; 13 September 2010.
URL: http://msdn.microsoft.com/en-us/library/system.net.sockets.socket.aspx.
Accessed 5 July 2011.

18 Catalyst Development. Blocking vs. non-blocking sockets [online].
Developerfusion; 2 October 2008.
URL: http://www.developerfusion.com/article/28/introduction-to-tcpip/8/.
Accessed 10 August 2011.

19 Tamir G. SharpSSH - A secure shell (SSH) library for .NET [online].
URL: http://www.tamirgal.com/blog/page/SharpSSH.aspx.
Accessed 31 March 2011.

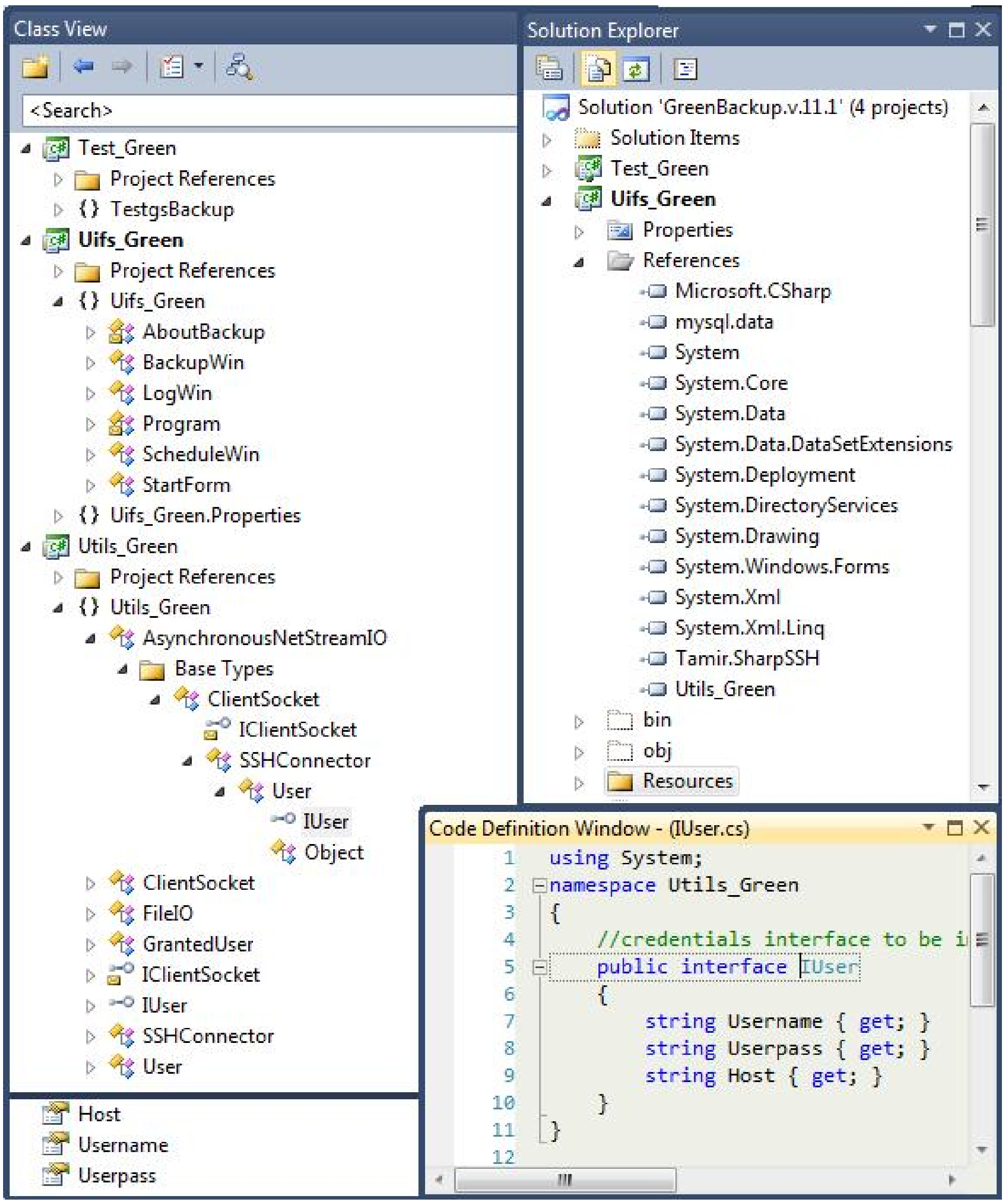
20 Akim D., Denny J.E., Bison P.E. GNU parser generator [online]. GNU Operating System.
URL: http://gnuwin32.sourceforge.net/packages/bison.htm.
Accessed 5 March 2011.

21 Microsoft Corporation. Using an asynchronous client socket [online]. MSDN Library.
URL: http://msdn.microsoft.com/en-us/library/bbx2eya8.aspx.
Accessed 20 July 2011.

22 Microsoft Corporation. Stream methods [online]. MSDN Library.
URL: http://msdn.microsoft.com/en-us/library/system.io.stream_methods.aspx.
Accessed 10 July 2011.

23 Richard B. C# network programming. 1151 Marina Village Parkway, Alameda: Sybex Inc.; 2003.

Appendix 1. Project Structure



Class View

<Search>

- Test_Green
  - Project References
  - {} TestgsBackup
- **Uifs_Green**
  - Project References
  - {} Uifs_Green
    - AboutBackup
    - BackupWin
    - LogWin
    - Program
    - ScheduleWin
    - StartForm
  - {} Uifs_Green.Properties
- Utils_Green
  - Project References
  - {} Utils_Green
    - AsynchronousNetStreamIO
      - Base Types
        - ClientSocket
          - IClientSocket
          - SSHConnector
            - User
              - IUser
              - Object
    - ClientSocket
    - FileIO
    - GrantedUser
    - IClientSocket
    - IUser
    - SSHConnector
    - User

Host
Username
Userpass

Solution Explorer

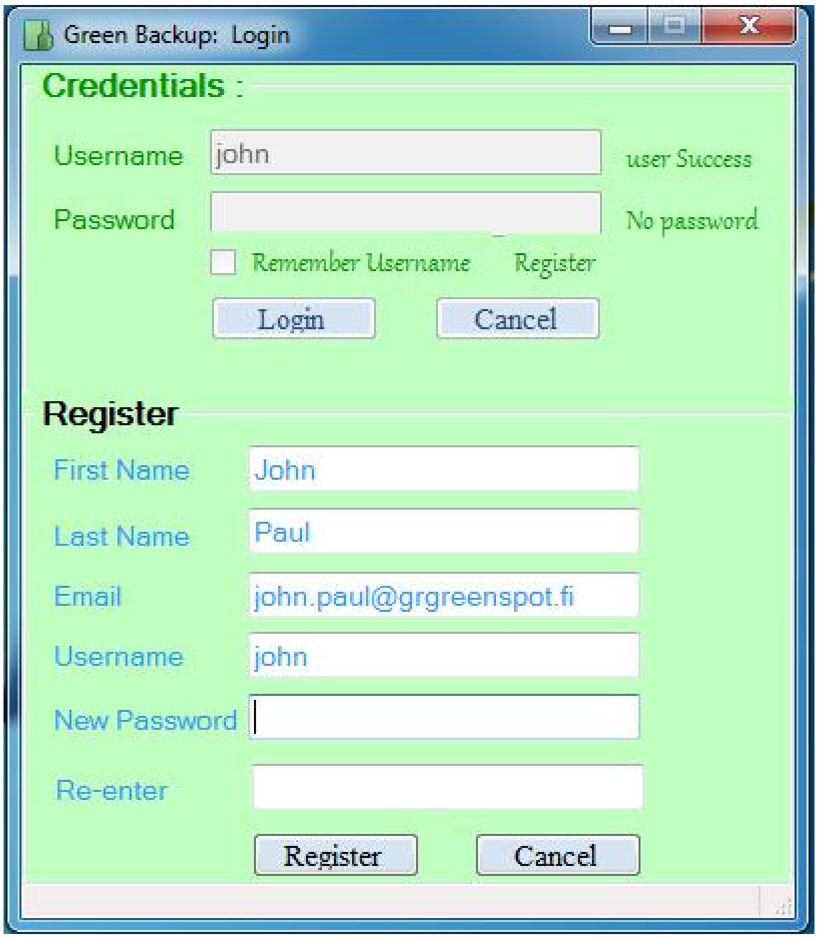
- Solution 'GreenBackup.v.11.1' (4 projects)
  - Solution Items
  - Test_Green
  - **Uifs_Green**
    - Properties
    - References
      - Microsoft.CSharp
      - mysql.data
      - System
      - System.Core
      - System.Data
      - System.Data.DataSetExtensions
      - System.Deployment
      - System.DirectoryServices
      - System.Drawing
      - System.Windows.Forms
      - System.Xml
      - System.Xml.Linq
      - Tamir.SharpSSH
      - Utils_Green
    - bin
    - obj
    - Resources

Code Definition Window - (IUser.cs)

```
using System;
namespace Utils_Green
{
    //credentials interface to be i
    public interface IUser
    {
        string Username { get; }
        string Userpass { get; }
        string Host { get; }
    }
}
```

Figure 12. Project structure

Appendix 2. Login and Register Window


Green Backup: Login

Credentials :

Username	john	user Success
Password		No password

Register

First Name	John
Last Name	Paul
Email	john.paul@grgreenspot.fi
Username	john
New Password	
Re-enter	

Figure 13. Login and register form

Appendix 3. Client User Class

```
// <Copyright file=" User.cs " Company=" Greenspot Backup">
// <Author> @Dawit Nida
// <Last Edited on>08-24-2011
// <Summary> Class representing a User.cs entity. </Summary>
using System;
using System.Data;
using System.Configuration;
using System.Windows.Forms;
using MySql.Data.MySqlClient;
namespace Utils_Green
{
    public class User : IUser
    {
        private readonly string _host = @"gsbackup.greenspot.fi";
        private string _name = string.Empty;
        private string _passwd = string.Empty;
        /// <summary>
        /// Default constructor with name and password
        /// </summary>
        /// <param name="name"></param>
        /// <param name="pass"></param>
        public User(string name, string pass)
        {
            _name = name;
            _passwd = pass;
            // _host = @"gsbackup.greenspot.fi";
        }
        public User()
        {
        }
        #region IUser Members
        /// <summary>
        /// create auto-implemented Properties
        /// </summary>
        public string Host
        {
            get
            {
                return _host;
            }
        }
        public string Username
        {
            get
            {
                return _name;
            }
            set
            {
                if (value != null)
                    _name = value;
                else
                    _name = string.Empty;
            }
        }
        // Do the same for password
        public string Userpass
        {
            get
            {
                return _passwd;
            }
            set
            {
                if (value != null)
                    _passwd = value;
                else
                    _passwd = string.Empty;
            }
        }
        /// <summary>
        /// Validate User check username->> return true if OK
        /// more validation todo
        /// </summary>
        /// <param name="input"></param>
        /// <returns></returns>
        public  bool IsValid(string input)
        {
            // check input is not null, then check if input has the required
            if (input.Length > 0)
            {
                string regexString = @"^[a-zA-Z\.\-_]{4,10}$";
                RegexStringValidator regexValidator = new RegexStringValida-
tor(regexString);

                try
                {
                    regexValidator.Validate(input);
                    return true;
                }
                catch (ArgumentException)
                {
                    return false;
                }
            }
            return false;
        }
        #endregion

        //validate user email address using regular expres-
sion:(Provided..msdn...regular expression
        public  bool IsValidEmail(string inputEmail)
        {
            string regexString = @"^[a-zA-Z\.\-_]+@([a-zA-Z\.\-_]+\.)+[a-zA-
Z]{2,4}$";
            RegexStringValidator regexValidator = new RegexStringValida-
tor(regexString);
            if (inputEmail.Length > 0)
            {
                try
                {
                    regexValidator.Validate(inputEmail);
                    return true;
                }
                catch (ArgumentException)
                {
                    return false;
                }
            }
            return false;
        }

        public void InsertNewUser(string fname, string lname, string email,string 
username,string passwd,string repasswd)
        {
            string connectionString = ConfigurationManag-
er.ConnectionStrings["Backup_Users"].ConnectionString;

            // create new connection using the connection string stored at con-
nectionString string above
            MySqlConnection connection = new MySqlConnection(connectionString);

            try
            {
                //open db connection using connection string
                connection.Open();

                string sqlInsert = " INSERT INTO GsBackupUsers VALUES 
(@fname,@lname,@email,@username,@passwd,@repasswd,@date);";
                MySqlCommand insertUser = new MySqlCommand(sqlInsert, connec-
tion);

                insertUser.Parameters.AddWithValue("@fname", fname);
                insertUser.Parameters.AddWithValue("@lname", lname);
                insertUser.Parameters.AddWithValue("@email", email);
                insertUser.Parameters.AddWithValue("@username", username);
                insertUser.Parameters.AddWithValue("@passwd", passwd);
                insertUser.Parameters.AddWithValue("@repasswd", repasswd);
                insertUser.Parameters.AddWithValue("@date", DateTime.Now);
                insertUser.ExecuteNonQuery();
                insertUser.Dispose();
                insertUser = null;

                //test case
                MessageBox.Show(String.Format("{0} {1} : has registered success-
fully. Thank you.",fname, lname), "Registration Info");
            }
            catch (MySqlException ex)
            {
                //throw exception if db connection is not okay
                 MessageBox.Show("Mysql Error. \n " + ex.ToString(), "Database 
Connection Info");
            }
            finally
            {
                //finally release all database resources
                if (connection != null)
                    connection.Close();
            }
        }
    }
}
```

Appendix 4. Client Socket Creator Class

```
// <Copyright file=" Client.cs" Company=" Greenspot Backup">
// <Author> @Dawit Nida
// <Last Edited on>08-24-2011
// <Summary> Class representing a Client.cs entity.</Summary>

using System;
using System.Net;
using System.Net.Sockets;
using System.Windows.Forms;
using System.Threading;

namespace Utils_Green
{
    public class ClientSocket : SSHConnector, IClientSocket
    {
        private IPEndPoint _endIP;
        private readonly IPAddress _address = IPAddress.Parse("83.143.217.188");
        private readonly int _portNumber = 20011;
        private Socket clientSock;
        private User loggedUser;


        /// <summary>
        /// constructor for ClientSocket Class
        /// </summary>
        /// <param name="clientSock"></param>
        /// <param name="user"></param>
        protected ClientSocket(Socket clientSock, User user)
            : base(user)
        {
            // member initialization
            _address = IP;
            _portNumber = Port;
            _endIP = new IPEndPoint(_address, _portNumber);
        }
        /// <summary>
        /// ClientSocket constructor overloaded
        /// </summary>
        /// <param name="onlineUser"></param>
        public ClientSocket(User onlineUser)
            : base(onlineUser)
        {
            loggedUser = onlineUser;
        }

        #region IClientSocket Members
        public IPAddress IP
        {
            get { return _address; }
        }
        public int Port
        {
            get { return _portNumber; }

        }
        public IPEndPoint Endpoint
        {
            get { return _endIP; }
            set
            {
                _endIP = value;
            }
        }

        #endregion
        /// <summary>
        /// create socket for the client and return Socket object to be used
        /// </summary>


        public Socket EstablishClientSock()
        {
            try
            {
                // create endpoint object using ip and port number
                Endpoint = new IPEndPoint(IP, Port);
                clientSock = new Socket(AddressFamily.InterNetwork,
                                    SocketType.Stream, ProtocolType.Tcp);

               clientSock.Connect(Endpoint);
                //test case
                MessageBox.Show(" Socket Connection Successful!", "Socket Connec-
tion Info");
                return clientSock;
            }
            catch (SocketException sockEx)
            {
                //test Connection
                MessageBox.Show(" Socket Connection Refused: Check Firewall Rules
\n Closing Socket Connection"
                                + sockEx.ToString(), " Socket Connection Info");
                return null;
            }
            finally
            {
                if (clientSock != null)
                    //close Socket connection safely
                    clientSock.Close();
            }
        }
    }
}
```

Appendix 5. Client File IO Class

```
// <Copyright file="FileIO.cs" Company=" Greenspot Backup">
// <Author> @Dawit Nida
// <Last Edited on>08-24-2011
// <Summary> Class representing a FileIO.cs entity.</Summary>

using System;
using System.Collections.Generic;
using System.IO;
using System.Windows.Forms;

namespace Utils_Green
{
    public class FileIO
    {
        /// <summary>
        /// List all ready drives and add it to the combo box
        /// </summary>
        /// <param name="listcbx"></param>
        public static void ListDrives(ComboBox listcbx)
        {
            // get all ready drives and save it into the array
            DriveInfo[] drivesInfo = DriveInfo.GetDrives();
            foreach (DriveInfo drive in drivesInfo)
            {
                if (drive.IsReady)
                {
                    listcbx.Items.Add(drive.Name);
                }
                else
                {
                    //test case
                    MessageBox.Show(" Drive " + drive.Name + " not ready!", "
Drive Selector Info");
                }
            }
        }
        /// <summary>
        /// get drives from the host computer and return drive name
        /// Add to the combobox list
        /// </summary>
        /// <param name="drivescbx"></param>
        /// <returns></returns>
        public static string GetSelectedDriveName(ComboBox drivescbx)
        {
            string dr = null;
            dr = drivescbx.GetItemText(drivescbx.SelectedItem);

            if (dr.Length > 0)
            {
                //Test case
                MessageBox.Show("Selected Drive " + dr, " Drive Selector Info");
                return dr;
            }
            else
            {
                drivescbx.SelectedIndex = -1;
                drivescbx.SelectedValue = drivescbx.SelectedIndex;
                dr = drivescbx.GetItemText(drivescbx.SelectedItem);

                //test case
                //   MessageBox.Show("Default drive" + dr);
                return @"D:\";
            }
        }
        /// <summary>
        /// 
        /// </summary>
        /// <param name="tree"></param>
        /// <param name="drive"></param>
        public void DisplayDirectoriesAndFiles(TreeView tree, string drive)
        {
            //clear everything from the treeview
            tree.Nodes.Clear();
            tree.BeginUpdate();
            //get all drives from the computer
            tree.Nodes.Add(Environment.UserName);

            DirectoryInfo di = new DirectoryInfo(drive);
            DirectoryInfo[] dirInfo = di.GetDirectories();

            foreach (DirectoryInfo rootDirs in dirInfo)
            {
                TreeNode parent = new TreeNode(rootDirs.FullName);

                parent.ImageIndex = 0;
                parent.Tag = rootDirs;
                tree.Nodes[0].Nodes.Add(parent);
                GetFiles(rootDirs, parent);
                tree.EndUpdate();
            }
        }
        /// <summary>
        /// 
        /// </summary>
        /// <param name="dirInfo"></param>
        /// <param name="child"></param>
        private void GetFiles(DirectoryInfo dirInfo, TreeNode child)
        {
            try
            {
                DirectoryInfo[] dirInfoArray = dirInfo.GetDirectories();
                if (dirInfoArray.Length != 0)
                {
                    foreach (DirectoryInfo subDirs in dirInfoArray)
                    {
                        //Add the subdirectories to the upper level node
                        TreeNode dirNode = new TreeNode();
                        dirNode = child.Nodes.Add(subDirs.Name);
                        GetFiles(subDirs, dirNode);
                        dirNode.Tag = subDirs;
                    }
                    // Get any files for this node.
                    FileInfo[] files = dirInfo.GetFiles();
                    // After placing the nodes place the files in that subdirec-
tory
                    foreach (FileInfo file in files)
                    {
                        TreeNode fileNode = new TreeNode(file.Name);
                        child.Nodes.Add(fileNode);
                    }
                }
            }
            catch (UnauthorizedAccessException)
            {
                //test case
                //  MessageBox.Show( ex.ToString() + "File Access denied " , "
File Access Info");
            }
        }
        // Testcase
        private static List<string> readyFiles = new List<string>();
        public static string StoreListedFiles(ListView box)
        {
            ListView.CheckedListViewItemCollection chked = box.CheckedItems;
            for (int i = 0; i < chked.Count; )
            {
                string filename = box.CheckedItems[i].Text;
                foreach (ListViewItem items in chked)
                {
                    items.Checked = true;
                    readyFiles.Add(filename); ;
                }
                return filename;
            }
            return null;
        }
    }
}
```

Appendix 6. Server Open Port List

```
~# nmap -sS -O 127.0.0.1
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2011-06-10 14:24
EEST
Stats: 0:00:03 elapsed; 0 hosts completed (1 up), 1 undergoing OS Scan
Interesting ports on localhost.localdomain (127.0.0.1):
Not shown: 1675 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
80/tcp   open  http
111/tcp  open  rpcbind
3306/tcp open  mysql
# Accept tcp packets on destination ports 6881-6890
~# iptables -A INPUT -p tcp --dport 20011 -j ACCEPT
```

Appendix 7. Server Socket Creator and File Transfer Class

```
// <copyright file="ClientSocketListener.cs" company=" Greenspot Backup">
// <author>Dawit Nida</author>
// <date>08-24-2011</date>
// <summary>Class representing a ClientSocketListener.cs entity.</summary>

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Runtime.Serialization;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;


namespace FileTransfer
{
    public class ClientSocketListener
    {
        private const int _portNumber = 20011;
        private const int backlog = 20;

        private IPEndPoint _remoteEndPoint = null;
        private Socket listenerSocket = null;
        private AsyncCallback _clientConnectCallback;
        private AsyncCallback _clientDisconnectCallback;

        // Thread signal.
        private static ManualResetEvent ConnectionDone = new ManualRe-
setEvent(false);

        #region IServerSocket Members
        public int Port
        {
            get { return _portNumber; }
        }
        public IPEndPoint RemoteEndpoint
        {
            get { return _remoteEndPoint; }
            set
            {
                _remoteEndPoint = value;
            }
        }
        #endregion

        public ClientSocketListener()
        {
            _clientConnectCallback = new Async-
Callback(this.OnClientSocketConnect);
            _clientDisconnectCallback = new Async-
Callback(this.OnClientSocketDisConnect);
        }
        public void AcceptSocketConnection()
        {
            // create socket object
            listenerSocket = new Socket(AddressFamily.InterNetwork, Socket-
Type.Stream, ProtocolType.Tcp);
            RemoteEndpoint = new IPEndPoint(IPAddress.Any, Port);

            try
            {
                // and bind the socket to the endpoint and wait for any incoming
connections...
                listenerSocket.Bind(RemoteEndpoint);
                // start an asynchronous socket to listen for any connections
                listenerSocket.Listen(backlog);
                Console.WriteLine(" Server State listening and waiting for Cli-
ent...");

                while (true)
                {
                    // set the event to nonsignaled state
                    ConnectionDone.Reset();

                    // accept an asynchronous socket from any connections
                    listenerSocket.BeginAccept(OnClientSocketConnect, listener-
Socket);

                    // wait at least for one client request
                    ConnectionDone.WaitOne();
                }
            }
            catch (Exception sockEx)
            {
                Console.WriteLine(" Asynchronous Socket could not be created. \n"
+ sockEx.ToString());
            }
        }

        public void OnClientSocketConnect(IAsyncResult asyncResult)
        {

            // get the created socket and use it
            Socket listener = (Socket)asyncResult.AsyncState;
            Socket tempSocket = listener.EndAccept(asyncResult);
            Console.WriteLine(" Client..." + tempSocket.RemoteEndPoint.ToString()
+ " connected.");

            // signal that the connection has been made.
            ConnectionDone.Set();

            FileHandler filehandler = new FileHandler(tempSocket);
            filehandler.StartRead();

        }

        public void OnClientSocketDisConnect(IAsyncResult asyncResult)
        {
            try
            {
                Socket tempSocket = (Socket)asyncResult.AsyncState;
                // sending the data to the client ends here
                tempSocket.EndDisconnect(asyncResult);
                Console.WriteLine(" Client..." + tempSock-
et.RemoteEndPoint.ToString() + " disconnected safely.");
                tempSocket.Shutdown(SocketShutdown.Both);
                tempSocket.Close();

            }
            catch (Exception sockEx)
            {
                Console.WriteLine(" Error disconnecting the client." +
sockEx.ToString());
            }
        }

        static void Main(string[] args)
        {
            ClientSocketListener listener = new ClientSocketListener();
            listener.AcceptSocketConnection();
            Console.ReadLine();
        }
    }

    [Serializable]
    public class FileHandler
    {

        private NetworkStream newStream = null;
        private Stream inputStream = null;
        private Stream outputStream = null;

        private static ManualResetEvent ReceiveEnded = new ManualRe-
setEvent(false);

        private byte[] buffer;

        private AsyncCallback _recieveCallack;
        private AsyncCallback _sendCallback;
        private AsyncCallback _fileBackupCompleteCallback;

        public FileHandler(Socket serverSocket)
        {
            buffer = new byte[1024 * 5];
            newStream = new NetworkStream(serverSocket);
            _recieveCallack = new AsyncCallback(this.OnReceiveComplete);
            _sendCallback = new AsyncCallback(this.OnDataSendComplete);
            _fileBackupCompleteCallback = new Async-
Callback(this.OnFileBackupComplete);
        }

        public void StartRead()
        {
            // check if netstream is readable
            if (newStream.CanRead)
            {
                newStream.BeginRead(buffer, 0, buffer.Length, _recieveCallack,
null);
                ReceiveEnded.WaitOne();
            }
            else
```

```
                {
                    Console.WriteLine(" NetworkStream is unreadable.");
                }
            }

            protected string SplitFileSizeName(string stringMessage, short i)
            {
                char splitter = ' ';
                string[] size = stringMessage.Split(splitter);
                return size[i];
            }

            private static int TryToParseFileSize(string inputMessage)
            {
                int size = 0;
                bool success = Int32.TryParse(inputMessage, out size);
                if (success)
                {
                    return size;
                }
                else
                {
                    Console.WriteLine(" File Size could not be retrieved.");
                    return 0;
                }
            }

            int receivedFileSize = 0;
            string receivedFileName = string.Empty;
            string receivedFile = string.Empty;
            string fullPath = string.Empty;

            private void OnReceiveComplete(IAsyncResult asycResult)
            {
                int bytesReceived = newStream.EndRead(asycResult);

                short i = 0;
                try
                {
                    if (bytesReceived > 0)
                    {
                        string receivedMessage = Encoding.ASCII.GetString(buffer, 0,
bytesReceived);
                        Console.WriteLine(" Received bytes {0} : with size {1} \n ",
receivedMessage, bytesReceived);

                        string location = @"/home/john";
                        i = 1;
                        receivedFileName = SplitFileSizeName(receivedMessage, i);

                        //create a new file under /home/username for read/write
                        outputStream = new FileStream(location + "//" + receivedFile-
Name, FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.ReadWrite);
                        string fullPath = location + "/" + receivedFileName;

                        i = 0;
                        receivedFileSize = TryToParse-
FileSize(SplitFileSizeName(receivedMessage, i));

                        // i = 2;
                        receivedFile = SplitFileSizeName(receivedMessage, i);
                        Console.WriteLine(" File Size '{0}' \t '{1}' \n", received-
FileSize, receivedFileName);

                        i = 0;
                        receivedFileSize = TryToParse-
FileSize(SplitFileSizeName(receivedMessage, i));

                        byte[] receivedData = new byte[receivedFileSize];

                        inputStream = new FileStream(location + "//" + receivedFile-
Name, FileMode.Open, FileAccess.ReadWrite, FileShare.ReadWrite);
                        Console.WriteLine("  File '{0}' opened for write.", full-
Path);

                        //Console.WriteLine(" Deserialize '{0}' bytes", received-
FileSize);
                        //IFormatter formatter = new BinaryFormatter();
                        //byte[] data = (byte[])formatter.Deserialize(inputStream);

                        Console.WriteLine(" Begin reading from network stream here:
Status  \n  Received File Size: '{0}' bytes Received File Name: '{1}' Received
Data Length: {2} bytes.", receivedFileSize, receivedFileName, receivedDa-
ta.Length);
                        inputStream.BeginRead(receivedData, 0, receivedFileSize,
_fileBackupCompleteCallback, null);

                        Console.WriteLine("  Cleaning Stream and '{0}' Closed here!",
fullPath);
                        //clean up everything
                        outputStream.Close();
                        outputStream.Dispose();

                    }
                    else
                    {
                        Console.WriteLine(" Data unavailable ....Network streaming
could not be acheived!");
                        newStream.Close();
                        newStream = null;
                    }
                }
                catch (Exception randomEx)
                {
                    Console.WriteLine(" Exception cought here. \n" + ran-
domEx.ToString());
                }
            }

            public void OnDataSendComplete(IAsyncResult asycResult)
            {
                newStream.EndRead(asycResult);
                Console.WriteLine(" Reading data from network stream is completed.
");
                //inputStream.BeginRead(buffer, 0, buffer.Length,
_fileBackupCompleteCallback, null);
            }

            private void OnFileBackupComplete(IAsyncResult asycResult)
            {
                int bytesRead = inputStream.EndRead(asycResult);

                // check if any data left on the stream
                if (receivedFileSize > 0)
                {
                    SaveFileToDisk(inputStream, outputStream, receivedFileSize);
                    Console.WriteLine(" File successfully saved.");
                }
                inputStream.Close();
                newStream.Flush();
                newStream.Close();
            }

            private void SaveFileToDisk(Stream inputStream, Stream outputStream, int
    SizeOfBuffer)
            {
                BinaryReader binReader = new BinaryReader(inputStream);
                BinaryWriter binWriter = new BinaryWriter(outputStream);
                Console.WriteLine(" Start saving file '{0}' to disk: Status Received
    File Size: '{1}' with size '{1}' bytes.",receivedFileName, receivedFileSize);

                //create a buffer to hold the bytes
                byte[] bufferReader = new Byte[SizeOfBuffer];

                try
                {
                    if (inputStream.CanRead)
                    {
                        Console.WriteLine(" \n Status: Received File Size '{0}' ",
    receivedFileSize);
                        if (receivedFileSize > 0)
                        {
                            Console.WriteLine(" Before Binary writer size '{0}' bytes
    ", bufferReader.Length);
                            binWriter.Write(bufferReader, 0, SizeOfBuffer);
                        }
                    }
                    else
                    {
                        //clean up everything
                        binWriter.Flush();
                        binReader.Close();
                        binWriter.Close();
                    }
                }
                catch (IOException ex)
                {
                    Console.WriteLine(" File writing failed. \n" + ex.ToString());
                }
            }

        }
    }
```