# PROJECT CATEGORY

This software will follow Object Oriented Programming Paradigm and use below mentioned areas:

OOP Language: C#, Typescript

RDBMS: SQLite

Networking: TCP/IP

Applications: Desktop Systems , Web Application , Personal Cloud

# Hardware and Software Specification

## For Mobile:

### Hardware Requirement

* **Phone Memory/SD Card Memory :** 300 MB of available hard disk space
* **RAM :** 128 MB or higher
* **Processor :** 484 MHz or faster

### Software Requirement

* Windows 7
* Ubuntu 12.04
* Android Honey Comb or Higher

## For Desktop

### Hardware Requirement

* Disk Space: 50 Mb of available disk space.
* RAM: 256 MB
* Processor: 1 GHz dual core

### Software Requirement

* Windows XP, Windows 07 Operating System.
* Microsoft .NET framework 3.5. / .NET framework 4
* Ubuntu
* Dia for Drawing and modeling
* Microsoft office word for documentation

# REQUIREMENTS AND ANALYSIS

## Problem Definition

### Existing System

We use different types of storage medium for different types of devices. Here are some Advantages of external storage:

* External storage devices provide additional storage other than that available in computer.
* Data can be transported easily from one place to another.
* It is useful to store software and data that is not needed frequently.
* External storage also works as data backup.
* This back up may prove useful at times such as fire or theft because important data is not lost.

How many storage devices anyone owns now? It will be at least more than ten. Do you know which of your device contains what content? The answer is mostly no. We need a centralized solution for storing information about the files we have. Cloud based storage is costly. Why can't we turn our own storages devices into a connected personal cloud system?  So we need to store information about the files we have and access them from the server whenever we need them. The existing system allows user to share their personal data on the web through various cloud based applications but they do not allow us to manage all our devices’ storages in one place. The existing ones, on the other hand, are not as user friendly as well. Sometimes they create confusion while logging in to the web server from other devices.

### our target

Using our application, the users can access information about the data of all his storage devices from one place with a very organized way. They can store and synchronize data of their desktop, laptop and/or smart phone. The main intension is to utilize personal storage devices, avoid duplicity of files in several devices, easy to find which storage device has the desired data.

### Work To Be Done

We need to implement several clients (Host App) and server as given below:

1. **Windows Host App**: Windows OS has an API called **FileSystemWatcher.** I will use FileSystemWatcher to watch for changes in a specified directory. You can watch for changes in files and subdirectories of the specified directory. You can create a component to watch files on a local computer, a network drive, or a remote computer.
2. **Ubuntu Linux Host App**: The QFileSystemWatcher class provides an interface for monitoring files and directories for modifications. QFileSystemWatcher monitors the file system for changes to files and directories by watching a list of specified paths. Call addPath() to watch a particular file or directory. Multiple paths can be added using the addPaths() function. Existing paths can be removed by using the removePath() and removePaths() functions. FileSystemWatcher examines each path added to it. Files that have been added to the QFileSystemWatcher can be accessed using the files() function, and directories using the directories() function. The fileChanged() signal is emitted when a file has been modified, renamed or removed from disk. Similarly, the directoryChanged() signal is emitted when a directory or its contents is modified or removed. Note that QFileSystemWatcher stops monitoring files once they have been renamed or removed from disk, and directories once they have been removed from disk.
3. **Android Host App**: FileObserver (android.os.FileObserver) Monitors files (using inotify) to fire an event after files are accessed or changed by by any process on the device (including this one). FileObserver is an abstract class; subclasses must implement the event handler onEvent(int, String). Each FileObserver instance monitors a single file or directory. If a directory is monitored, events will be triggered for all files and subdirectories inside the monitored directory. An event mask is used to specify which changes or actions to report. Event type constants are used to describe the possible changes in the event mask as well as what actually happened in event callbacks.
4. **FMS Server / Controller**: This will be implemented using CodeIgniter framework and php. This will follow MVC architecture and expose REST interface so that data can be updated and synced from multiple and various kinds of operating systems.
5. **Web Viewer**: This will be implemented in HTML 5 and bootstrap which will allow users to view file information.

### Technical specification

#### For mobile app

* **Front End/ GUI Tools:** XAML (Extensible Application Markup Language) , QML , XML
* **IDE:** Visual Studio Express 2012 , Eclipse ,
* **Framework:** Microsoft .NET 4.0
* **Database:** Sqlite.
* **SDK:** Android SDK 8.0.
* **Cloud Technology:** SkyDrive , Google Drive

#### For desktop app

* **Front End/ GUI Tools:** Windows Presentation Framework (WPF)
* **IDE:** Visual Studio 2010
* **Framework:** Microsoft .NET 4.0
* **Database:** MySQL
* **Database Tool:** MySQL workbench CE
* **Operating Systems**: Windows XP, Windows 7
* **Cloud Technology**: Google Drive, Dropbox, SkyDrive

# Scope of the Solution

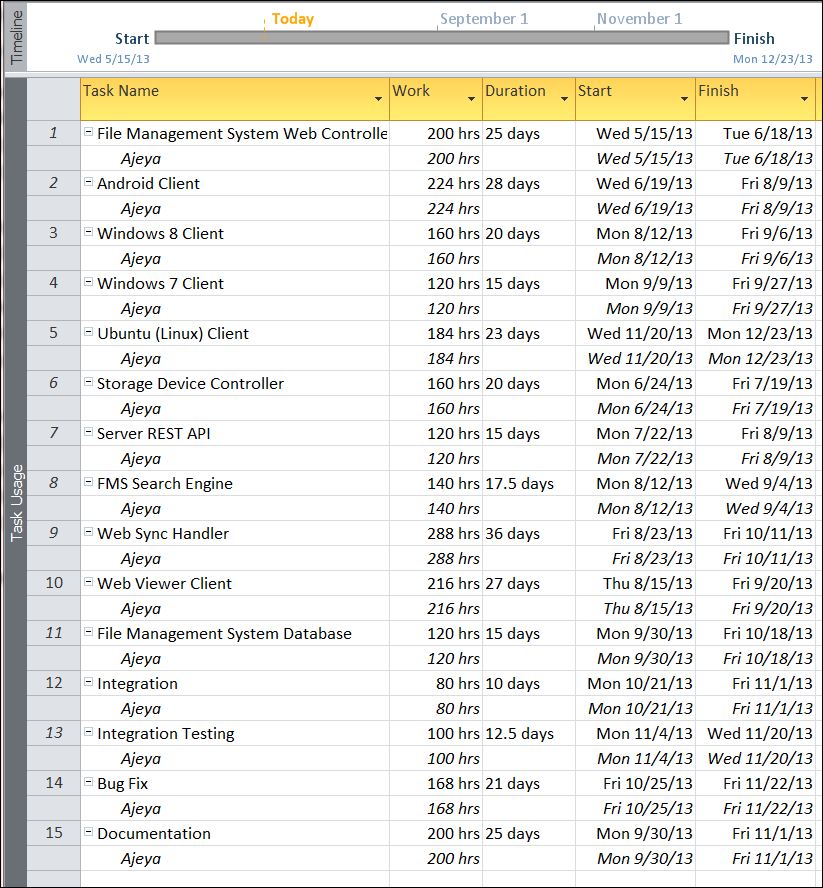
1. This app will not show or store the actual file. It will just display the file information. Then if the user wants to open the file it will use system default application to open the file.
2. This app won’t work in Windows Phone and Apple iPhone as these file system allow third party apps to access file information.
3. This app can copy, store and share file on explicit user request.

# Database & Table Details

The database used for this software is called f**msdb**. A screenshot from the MySQl workbench is given below. It shows the tables and its columns. The first row is the primary key.

|  |  |
| --- | --- |
| **Table** | **Column** |
| UserInfo | **userId**; userName; deviceOwned; files; |
| FileBackupInfo | **backupId**; backupDate; usedHosts; sourceDevices; files; |
| FileStreamInfo | **streamId**; streamDate; usedHost; sourceDevice; destinationIP; files; |
| StorageDeviceInfo | **deviceId**; lastScannedDate; lastUsedHost; files; filesBackupHere; |
| HostAppDeviceInfo | **hostDeviceId**; storageDevicesAttached; filesStreamed; type; |
| FileInfo | **fileId**; fileName; filePath; device; hostAppId; fileSize; fileType; creationDate; modifyDate; backups; streams; |
| FMSServerInfo | **serverId**; storageDevicesAttached; hostDevicesAttached; filesStreamed; fileBackedup; |

## Estimation



## Data Structure

|  |
| --- |
| UserInfo |
| public class UserInfo  {  public string userId;  public string userName;  public List< StorageDeviceInfo> deviceOwned;  public List<FileInfo> files;  } |

|  |
| --- |
| FileBackupInfo |
| public class FileBackupInfo  {  public string backupId;  public DateTime backupDate;  public List< HostAppDeviceInfo > usedHosts;  public List< StorageDeviceInfo >sourceDevices;  public List<FileInfo> files;  } |

|  |
| --- |
| FileStreamInfo |
| public class FileStreamInfo  {  public string streamId;  public DateTime streamDate;  public HostAppDeviceInfo usedHost;  public StorageDeviceInfo sourceDevice;  public string destinationIP;  public List<FileInfo> files;  } |

|  |
| --- |
| StorageDeviceInfo |
| public class StorageDeviceInfo  {  public string deviceId;  public DateTime lastScannedDate;  public HostAppDeviceInfo lastUsedHost;  public List<FileInfo> files;  public List<FileBackupInfo> filesBackupHere;  } |

|  |
| --- |
| HostAppType |
| public enum HostAppType  {  Android,  Win8,  Win7,  ubuntu  } |

|  |
| --- |
| HostAppDeviceInfo |
| public class HostAppDeviceInfo  {  public string hostDeviceId;  public List<StorageDeviceInfo> storageDevicesAttached;  public List<FileStreamInfo> filesStreamed;  public HostAppType type;  } |

|  |
| --- |
| FileInfo |
| public class FileInfo  {  public string fileName;  public string filePath;  public StorageDeviceInfo device;  public string hostAppId;  public string fileSize;  public string fileType;  public string creationDate;  public string modifyDate;  public List<FileBackupInfo> backups;  public List<FileStreamInfo> streams;  } |

|  |
| --- |
| FMSServerInfo |
| public class FMSServerInfo  {  public string serverId;  public List<StorageDeviceInfo> storageDevicesAttached;  public List<HostAppDeviceInfo> hostDevicesAttached;  public List<FileStreamInfo> filesStreamed;  public List<FileBackupInfo> fileBackedup;  } |

## Implementation Methodology

* Object Oriented Programming methodology will be adopted and c sharp will be used as programming language.
* .NET framework will be used to develop desktop
* Android SDK will be used for mobile app.
* User interface development will be done using WPF (Windows Presentation Foundation).
* Relational DBMS MySQL will be used to implement & execute SQL query to database.
* Agile Software Development model will be used while developing this software.