International Journal of Computer Applications (0975 – 8887)
Volume 186 – No.81, April 2025

Shared Media Web Service Architecture Design

Waseem I. Bader
Al-Salt College for Human Sciences,
Al-Balqa Applied University, Al-Salt, Jordan

ABSTRACT

Dealing with the media is one of the main subjects that developers need to handle when building their websites, projects and their related mobile applications. It can be quite challenging especially if these projects are developed using different programming languages and development environments. It can also become a struggle when these systems are applied within load-balancing environments or limited-sized archiving solutions.

At the same time, organizing the related websites and projects to share a common media service, can greatly improve the capability of administrating them with ease, keeping the consistency between different media files and the multi-system environment that uses them, reducing the time and effort needed for adding media capabilities to newer projects, or even making any desired changes for the media service functionality without affecting any project that uses it.

In this paper, a proposed shared media web service architecture design for a related group of projects is presented whichuses a single media web service to support all applications, providing all the functionality needed to ensure consistency between them while at the same time reducing the time and effort needed for programmers and developers to maintain them in the best possible way.The code samples in this paper were implemented using the ASP.net programming languagebut the same architecture can be implemented in any other development language and environment.

Keywords

Web Development, Media Management, Web Services.

1. INTRODUCTION

There are many organizations around the world that need to have multiple related websites, projects or even mobile applications at the same time[1], these multi-systems need has urged developers to come up with solutions regarding the code redundancy problems among other issues. But one of the major challenges that accompanies such systems is the Media management architecture.

In the simplest, and we might say most problematic approach, would be to make every project independent of the other projects regarding its handling with its media. This approach as direct as it can be, will force the developers to deal with big and challenging problems on the long run.

Take for an example, a university that has its own Student Registration website and another ELearning portal that deals with the same list of students. At the same time, the university has an Employee portal that provide services for the academic and administrative staff of the university. In addition to these systems, the university has a mobile application that provide exactly the same services for its members.

The Registration website for example allows the students to upload their profile pictures to it, and the ELearning portal would show these profile pictures and might allow to upload home works as well. Regarding the Employee portal, the Teachers can also provide profile pictures, and in the ELearning system, the class would show the profile picture of its instructor as well. All these media files can also be uploaded or viewed within the mobile application of the university.

If each system handles its own media files separate from the others, then the same media files uploaded to one system must be re-uploaded to the other systems as well, or duplicate media files must be shared across different systems which would come up with its own consistency problems[2].

An Architecture of a shared media web service is then the best solution to support all related projects to overcome the challenges aroused by the need to share media files between them.This shared media web service ispresented and discussed in details to suggest the best techniques that can be implemented to provide as much consistency, flexibility and effectivity as possible for developers in such a development architectural scenario, as shown in the following figure:

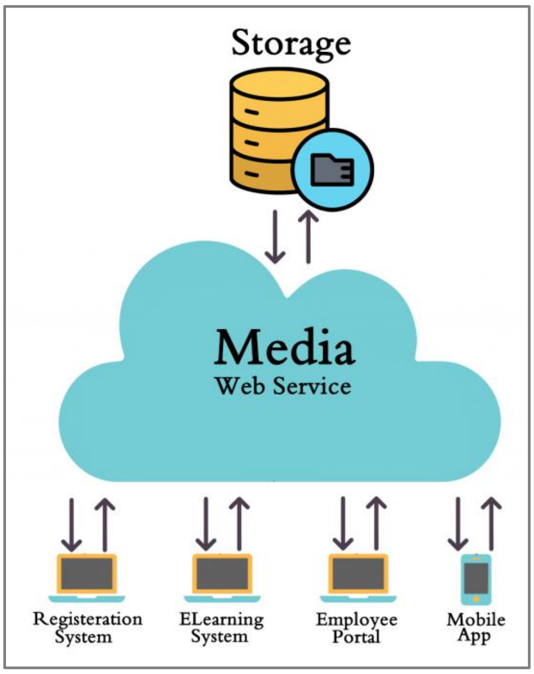


Figure 1: Shared MediaWebService Architecture

2. WEB SERVICE ARCHITECTURE

First of all, a web service is a piece of software that makes itself available over the internet and uses a standardized XML messaging system to provide a method of communication between different applications that might use different programming languages or development environments[3]. It can be used to provide the needed media processingtools among related websites by providing the shared services and media between them.

A web service is basically a web application which is generally a class containing a list of web methods that could be used by other applications. The web service should be deployed on a web server with a specific URL address in order for other applications to be able to reach the web service and consume its available web methods. In case of a web service built with ASP.net, it will have a URL similar to the following:

http://www.myUniversityDomain.com/myUniversityWebServices/UniversityWebService.asmx

Each supported functionality that the web service should provide will be presented as separate web method that any external application can access by passing the appropriate parameters to the web method and receiving the corresponding return value from the web service to the accessing application.The following figure shows the structure of a UniversityMedia Web Service with a sample web method calledDeleteFile that receives a String parameter that corresponds to the filename to be deleted from the media server, and then the method would return a Boolean value corresponding to a successful delete operation or a failed one as shown in the same figure:

```
namespace myUniversityWebServices
{
  [WebService(Namespace = "http://www.myUniversityDomain.com
")]
public class UniversityWebServices:
System.Web.Services.WebService
   {
      [WebMethod]
      public BooleanDeleteFile(String fileName)
      {
        Boolean fileDeleted = …
//the code to save the file from byte array format onto the server
         return fileDeleted;
      }
   }
}
```

Figure 2: University Web Service Structure

2.1Consuming the Web Service from different websites or projects.

In order to use the web service from an ASP.net website, a reference to the Media Web Service must be added in Visual Studio by using the URL address of the desired web service[4]. By right clicking the project in the solution explorer and choosing"Add Service Reference" option from the context menu as shown in the following figure:

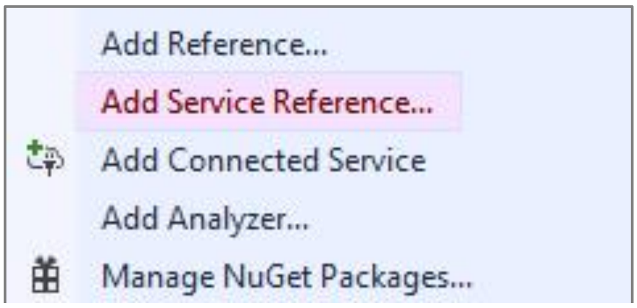
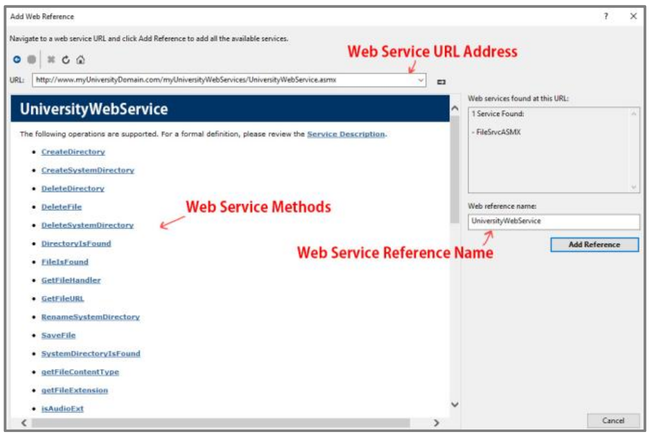


Figure 3: Adding a reference to a web service in Visual Studio

In the **Advanced** section of the **Add Service Reference** Window, the **Add Web Reference** button should be used and the web service URL address and reference name in the website should be set before adding the web service reference as shown in the following figure:



Add Web Reference

Navigate to a web service URL and click Add Reference to add all the available services.

URL: http://www.myUniversityDomain.com/myUniversityWebServices/UniversityWebService.asmx

UniversityWebService

The following operations are supported. For a formal definition, please review the Service Description.

- CreateDirectory
- CreateSystemDirectory
- DeleteDirectory
- DeleteFile
- DeleteSystemDirectory
- DirectoryIsFound
- FileIsFound
- GetFileHandler
- GetFileURL
- RenameSystemDirectory
- SaveFile
- SystemDirectoryIsFound
- getFileContentType
- getFileExtension
- isAudioExt

Web services found at this URL:
1 Service Found:
- FileSrvcASMX

Web reference name: UniversityWebService

Figure 4: Adding a reference to a web service in Visual Studio

After adding the web service reference in the ASP.net website project, the reference name set in the previous window will be used to access the web service methods in the required web pages of the web site as shown in the following example:

```
UniversityWebServiceuniws = new UniversityWebService ();
Boolean result = uniws.DeleteFile ("profilepic1.jpeg");
//deal with returned Boolean value here
…
```

Figure 5: using DeleteFile Web Service Method from the website

2.2Consuming the Web Service from the Mobile Application

To use the web service from an Android mobile application for example, any Soap supporting build-in or third-party libraries can be used. For Example, **KSoap2**is a lightweight open source library that can be usedto interoperate with most popular SOAP engines and hence can be used to access the employee web service in hand.**[5]**

In order to access a web method in the web service a **SoapObject** instance must be created with the web service URL address, method name and its corresponding parameter names and values, in addition to the use of an **HttpTransport** and **SoapSerializationEnvelope**objects as showing in the following code sample:

```
SoapObject soapRequest = new
SoapObject("http://www.myUniversityDomain.com/", "DeleteFile");

//Adding the corresponding parameters to the Soap request
PropertyInfo pi = new PropertyInfo();
pi.setName("fileName");
pi.setValue("profilepic1.jpeg ");
pi.setType(String.class);
soapRequest.addProperty(pi);

SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(
SoapEnvelope.VER11);

envelope.dotNet = true;
envelope.setOutputSoapObject(soapRequest);

HttpTransportSE httpTransport = new
HttpTransportSE("http://www.myUniversityDomain.com/myUniversityWebSer
vices/UniversityWebService.asmx");

Boolean result = false;
try {
    httpTransport.call("http://www.myUniversityDomain.com/DeleteFile",
envelope);
result= envelope.getResponse();
} catch (Exception exception) {
// Deal with any Exceptions here
        }
…
```

Figure 6: Accessing a web service method from an Android Application

After receiving the response from the web service method, the application would show convenient outputs to the user interface.

2.3Web Service Media Upload Method

The main operation that need to be available in the Media Web Service is that ability to upload files from the different websites and projects. In this operation,file content is sent to the web service for processing. There are two common techniques to send the content of the file to the web service method:

- Byte Arrays.
- Base64 Strings

When using Byte Arrays the actual content of the file is sent as a collection of bytes, where each byte contains 8 bits (binary numbers)**[6]**. As shown in the following figure:

```
[WebMethod]
public Boolean UploadFile (String fileName, byte[] bytes)
{
      String DestFolder = "PhysicalUploadFolderPath";
      Boolean fileUploaded = false;
//the code to save the file from byte array format onto the server
      string filePath = DestFolder + fileName;
      try
      {
File.WriteAllBytes(filePath, bytes);
fileUploaded = true;
      }
      catch ( System.Exeption exp )
      {
            //log exception
      }
       return fileUploaded;
}
```

Figure 7: Employee Web Service Upload File Web Method

On the other hand,Base64 (also known as tetrasexagesimal) is a group of binary-to-text encoding schemes that transforms binary data into a sequence of printable characters, limited to a set of 64 unique characters. More specifically, the source binary data is taken 6 bits at a time, then this group of 6 bits is mapped to one of 64 unique characters. with all binary-to-text encoding schemes, Base64 is designed to carry data stored in binary formats across channels that only reliably support text content, Base64 is quite easier for humans to read, as well as for applications to generate and parse, most programming languages provide different libraries for encoding and decoding binary data to and from Base64 Strings.**[7]**

After receiving the Base64 String, the web service then would decode the content to the actual byte array that would be later stored into the media server as shown in the following figure:

```
[WebMethod]
public Boolean UploadFile (String fileName, String base64Content)
{
byte[] fileData = Convert.FromBase64String(base64Content);
      Boolean fileUploaded = …
//the code to save the file from byte array format onto the server
        return fileUploaded;
    }
  }
```

Figure 8: Employee Web Service Upload File Web Method

Base64 String technique is better to use instead of the byte array one for different reasons:

- Some systems and applications only support text data. Base64 allows binary data to be included in such systems without compatibility issues
- Base64-encoded data can be easily included in URLs, JSON, XML, or other text-based formats without worrying about issues like special character encoding.
- Many protocols (like SMTP for email) are designed to handle text and may not support raw binary data. Using Base64 allows binary data to be sent over these protocols without issues
- Sending binary data directly can lead to corruption, especially if the data contains byte sequences that might be interpreted as control characters or delimiters.**[8]**

Now for Example, if we have a page in the registration website that allows students to upload their profile picture as shown in the following figure:

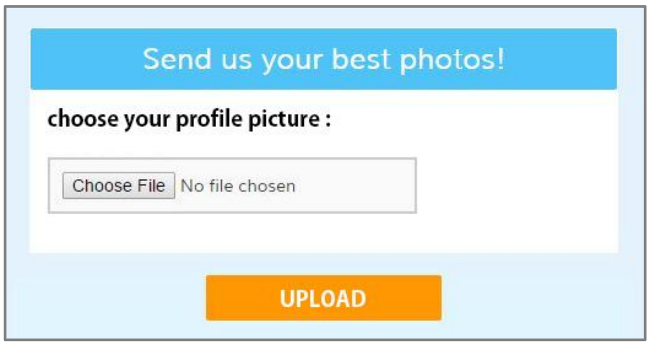


Figure 9: Registration Profile Picture Upload Form

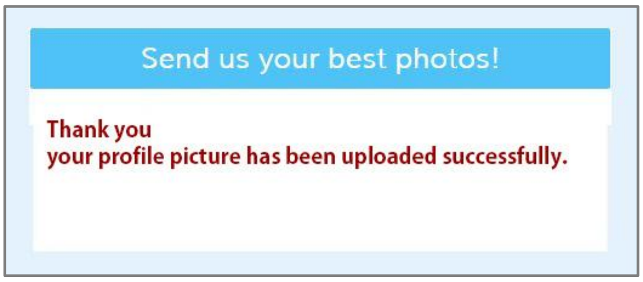
The content of the file is sent to the web service as Base64 String for processing and storing on the Media Server, and then a Boolean result is returned back to the website to notify the user of the result of the upload.

The Following code sample shows how to usethe**UploadFile** web method from the consuming website:

```
HttpPostedFile postedFile = myFileUpload.PostedFile;
byte[ ] fileBinaryData;
using (BinaryReader reader = new
BinaryReader(postedFile.InputStream))
{
        fileBinaryData =
reader.ReadBytes((int)postedfile.InputStream.Length);
}
String fileBase64String = Convert.ToBase64String(fileBinaryData);
UniversityWebServiceuniws = new UniversityWebService ();
Boolean result = uniws.UploadFile
("profilepic1.jpeg",fileBase64String);
//deal with returned Boolean value here
…
```

Figure 10: using UploadFile Web Service Method from the website

Based on the returned value from the web service, the user should be notified with a corresponding display message as follows:



Send us your best photos!

Thank you
your profile picture has been uploaded successfully.

Figure 11: UploadFile response display message to the user

3. DISPLAYING FILES FROM THE MEDIA WEB SERVICE

After uploading files to the media server, the web service need to provide ways to allow the different websites and projects to view the uploaded files. Two main techniques should be supported to provide full capabilities to the service consumers.

- Media Viewer Page and its URL Generator Web Method
- Media Web Handler and its URL Generator Web Method

3.1 Media Viewer Page and its URL Generator Web Method

In this technique, a special Media Viewer Page is prepared on the media web service project to view the different media files, and a Web Method is designed to create the exact link to the media in the viewer file.

The Media Viewer Page (ASPX page) would accept a Query String to the desired media file name and the file would be processed within the page and displayed to the user**[9]**.

In this technique two notes must be taken into consideration, the first one is that the web service should handle the media file name in an encrypted matter in order not to show the exact filename that is being shown. This is extremely important specially if the media file names contain numeric sequences that a hacker could change manually and display files he is not supposed to see. For example, if the file name is student1.pdf for example, a hacker can try to display information for student2.pdf if the desired file name is not encrypted before being displayed.

The second note, is that the service should display the media file in the viewer page without displaying any details regarding the physical location of the files. These notes are discussed further in the following code samples.

First a URL Generator web method should be added to the web service, this method will accept one parameter which is the filename and returns to the consuming project the URL to the MediaViewer ASPX page with the encrypted filename, this URL can be used in the website or project to link its users to the media file in the Media Viewer.

```
[WebMethod]
public String GetMediaFileURL(String FileName)
{
FileName =
System.Web.HttpUtility.UrlEncode(MyEncrypt(FileName));
return
"http://www.myUniversityDomain.com/myUniversityWebServices/M
ediaViewer.aspx?filename="+ FileName;
}
```

Figure 12: using GetMediaFileURL Web Service Method from the website

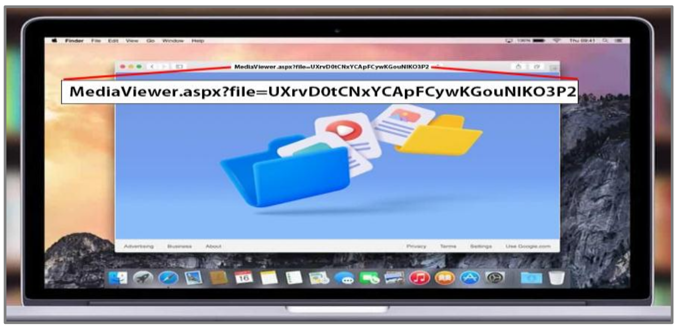
Note that MyEncrypt is a user-defined method that should be used to encrypt the filename in order to append it to the query string of the generated Media File URL.

Regarding the UrlEncode method it is a built-in method that converts characters that are not allowed in a URL such as blanks and punctuations into character-entity equivalents.

So when a project uses the **GetMediaFileURL** web method to get the URL of a file like student1.jpeg the result would look something like this:

http://www.myUniversityDomain.com/myUniversityWebServices/MediaViewer.aspx?filename=UXrvD0tCNxYCApFCywKGo

When the user clicks this generated link, the Media Viewer ASPX page is called and the encrypted filename is passed as a parameter to it.



MediaViewer.aspx?file=UXrvD0tCNxYCApFCywKGouNIKO3P2

Figure 13: using GetMediaFileURL Web Service Method from the website

The Media Viewer page then receives the encrypted file name query string parameter, decodes it, and displays the specified content in the web browser as shown in the following figure:

```
<html>
<body>
<formid="form1"runat="server">
<div>
        Welcome to File Viewer!
<%
           String DestServer = "d:/Uploads/";

           String filename = "" + Request.QueryString["filename"];

           filename =
MyDecrypt(System.Web.HttpUtility.UrlDecode(filename));


              Response.Clear();

              String fileFullPath = DestServer + filename;

if (System.IO.File.Exists(fileFullPath))
              {
                 Response.ContentType =
getFileContentType(getFileExtension(filename));
                 Response.WriteFile(fileFullPath);
                 Response.End();
              }
else
              {
                 Response.Write("File Not Found.");
              }
           }
%>
</div>
</form>
</body>
</html>
```

Figure 14: Media Viewer ASPX Page

As shown in the previous figure, the Media Viewer gets the filename from the Query String, URL decodes it, decrypts it back to its original value, then appends the physical location of the uploads folder to the filename and displays the content of the file based on the content type of its extension to the client. Note that using the **WriteFile** method from the response, avoids the displaying of the actual physical location of the media file in the resulting output.

To use this technique from the consumer part, the GetMediaFileURL is called to get the correct URL of the media file and a link is created to the generated Media Viewer URL to allow the user to view the file as shown in the following figure:

```

<%
UniversityWebServiceuniws = new UniversityWebService ();
String mediaURL = uniws.GetMediaFileURL ("profilepic1.jpeg");
%>

<a target=”_blank” href=”<%= mediaURL%>”>Click Here To View
The Profile Picture</a>
…
```

Figure 15: using GetMediaFileURL Web Service Method from the website

The user would see on his browser a link that when clicked will open the media file in the MediaViewer ASPX page as shown in figure:

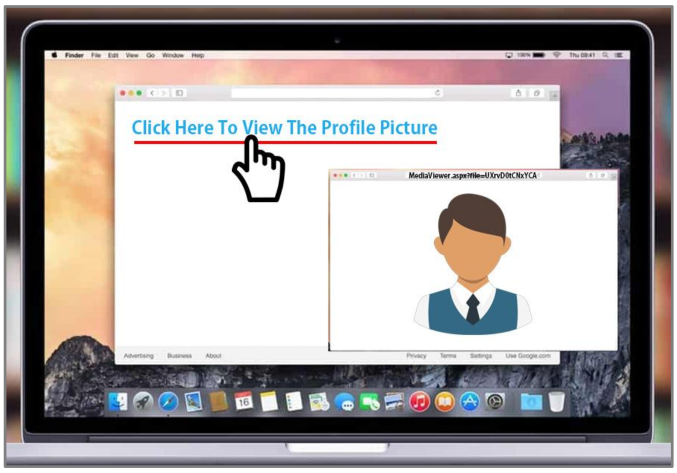

Click Here To View The Profile Picture
MediaViewer.aspx?file=UXrvD0tCNxYCA

Figure 16: the output of the GetMediaFileURL sample code

3.2Media Web Handler and its URL Generator Web Method

This second technique provides for the media web service consumers the ability to view the media content through their pages such as images or videos. As the first technique, a web method must be created to generate the URL of the desired media file, in addition to providing a web handler (ASHX) page to return the content of the media**[10]**.

The URL generator method is similar to the one used in the first technique, but the return URL will point the web handler that will be used to handle the media content as shown in the following figure:

```
[WebMethod]
public String GetMediaFileHandlerURL(String FileName)
{
FileName =
System.Web.HttpUtility.UrlEncode(MyEncrypt(FileName));

return
“http://www.myUniversityDomain.com/myUniversityWebServices/M
ediaHandler.ashx?filename="+ FileName;
}
```

Figure 17: using GetMediaFileHandlerURL Web Service Method from the website

Notice that we also want to encode and encrypt the filename in the generated URL.

The ASHX Media Web Handler is also similar to Media Viewer File in the first technique, but instead of viewing the content in the ASPX page, it returns the content for displaying to the consumer project, as shown in the figure:

```
publicclassMediaHandler : IHttpHandler
   {

publicvoid ProcessRequest(HttpContext context)
      {
String DestServer = "d:/Uploads/";
         String filename = "" +
context.Request.QueryString["filename"];
         filename =
MyDecrypt(System.Web.HttpUtility.UrlDecode(filename));
String fileFullPath = DestServer + filename;

if (System.IO.File.Exists(fileFullPath))
            {
                String fileExt = getFileExtension(fileFullPath);

if (isImgExt(fileExt))
                {
                    String contentType = getFileContentType(fileExt);
                    System.Drawing.Imaging.ImageFormat imgFormat =
FileSrvc.FileSrvcFunctions.getImageFormat(fileExt);

                    context.Response.ContentType = contentType;

                    System.Drawing.Image oImg =
System.Drawing.Image.FromFile(fileFullPath, true);
                    oImg.Save(context.Response.OutputStream,
imgFormat);
                    oImg.Dispose();
                }
elseif (isVideoExt(fileExt))
                {
                    String contentType = getFileContentType(fileExt);
                    context.Response.ContentType = contentType;
                    context.Response.WriteFile(fileFullPath);
                }
            }
        }
    }
}
```

Figure 18: the Media Handler code

Note that the isImgExt and is VideoExt are user-defined methods that check if the media extension is one of the images extensions such as jpg,jpeg,png,gif or one of the video extensions such as mp4,avi,mpeg. Likewise, the getFileExtension and getFileContentType as also user-defined methods that extracts the extension part of the file name and gets the corresponding content type based on it.

To use this technique from the consumer part, the GetMediaFileHandlerURL is called to get the correct URL of the media file and the generated Media Handler link is used to view the media in the project to the user as shown in the following figure:

```
<%
UniversityWebServiceuniws = new UniversityWebService ();
String mediaURL = uniws.GetMediaFileHandlerURL
("profilepic1.jpeg");
%>
<img src="<%= mediaURL%>"/>
…
```

Figure 19: using GetMediaFileHandlerURL Web Service Method from the website

The user would see on his browser the desired media file from the content returned by the Media Handler ASHX page as shown in figure:

Figure 20: the output of the GetMediaFileHandlerURL sample code

4. ENHANCING THE MEDIA WEB SERVICE

Many features can be added to the base media service type discussed in the previous parts, of these two major features are discussed to enhance the functionality and security of the web service which are:

- Arranging Media Service Types
- Providing Consumer Authentication

4.1 Arranging Media Service Types

In the previous parts, all the media files uploaded by the different media web service consumers are expected to be found in exactly the same location, which might be applicable for some cases specially if the number of consumers is few or the number of expected media files is limited, but in most cases this scenario is not the desired one, and an arrangement of the uploaded media files in sub locations are much needed. To achieve such arrangement, a service type must be set when processing the media files in the web service.

First, a Database Table must be created to store the service type information for each required service in the logic architecture. For each service we store a unique service ID, descriptive name, the physical sub location name, and a String Keyword that we'll be using in later parts, as shown in the following figure:

SID	SName	SDirectory	SKey
1	Registration System	REG_FOLDER	REG
2	ELearning System	ELEARN_FOLDER	ELEARN
3	Employee Portal	EMP_FOLDER	EMP
4	…	…	…

Figure 21: Media Services Database Table Sample

These services will have its own physical sub folder within the media service storage location, as shown in the following figure:

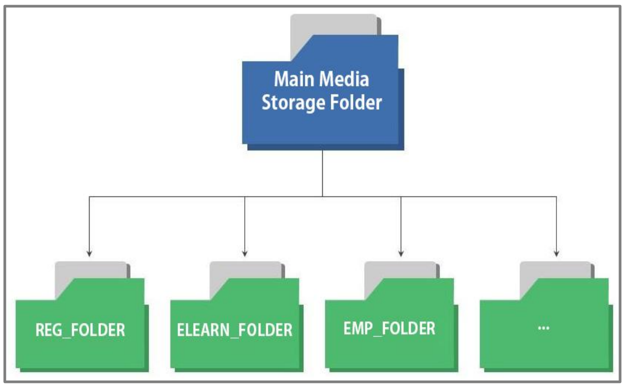


Figure 22: Media Service Storage Sub Systems Architecture

This Service type must be provided by the consumer projects when accessing the different web methods of the media web service. The **UploadFile** web method for example, must be modified to accept a service type in addition to the filename and file content. This service type is then used to get the corresponding sub folder of the service, to upload the media file to the correct sub folder in the media storage architecture as shown in the following code sample:

```
[WebMethod]
public Boolean UploadFile (int ServiceTypeID, String fileName, String
base64Content)
{
      String DestFolder = "PhysicalUploadFolderPath";
      String subDIR = //get the Directory that matches the service type ID from
DB
      Boolean fileUploaded = false;
      //the code to save the file from byte array format onto the server
      string filePath = DestFolder + subDIR+ fileName;
      try
      {
            File.WriteAllBytes(filePath, bytes);
            fileUploaded = true;
      }
      catch ( System.Exeption exp )
      {
            //log exception
      }
       return fileUploaded;      }
   }
```

Figure 23: Upload File Web Method with Service Types.

Likewise, the GetMediaFileURL would also accept the service type ID in addition to the filename, and both parameters are passed to the Media Viewer Page as shown in figure:

```
[WebMethod]
public String GetMediaFileURL(int ServiceTypeID, String FileName)
{
String ServiceTypeIDEnc =
System.Web.HttpUtility.UrlEncode(MyEncrypt(ServiceTypeID));
FileName =
System.Web.HttpUtility.UrlEncode(MyEncrypt(FileName));

return
"http://www.myUniversityDomain.com/myUniversityWebServices/M
ediaViewer.aspx?ServiceTypeIDEnc="+
ServiceTypeIDEnc+"&filename="+ FileName;
}
```

Figure 24: GetMediaFileURL Web Service Method with Service Types

The Media Viewer ASPX page as well will use now the received Service Type ID to get the appropriate Service Sub Directory to view the correct media file as shown in the following figure:

```
<html>
<body>
<formid="form1"runat="server">
<div>
      Welcome to File Viewer!
<%
        String DestServer = "d:/Uploads/";

String ServiceTypeIDEnc = "" +
Request.QueryString["ServiceTypeIDEnc"];

int ServiceTypeID =
MyDecrypt(System.Web.HttpUtility.UrlDecode(ServiceTypeIDEnc))
;

String subdir = //get the Service Directory that matches the service
type ID

        String filename = "" + Request.QueryString["filename"];

        filename =
MyDecrypt(System.Web.HttpUtility.UrlDecode(filename));

          Response.Clear();

          String fileFullPath = DestServer + subdir + filename;

if (System.IO.File.Exists(fileFullPath))
          {
            Response.ContentType =
getFileContentType(getFileExtension(filename));
            Response.WriteFile(fileFullPath);
            Response.End();
          }
else
          {
            Response.Write("File Not Found.");
          }
        }
%>
</div>
</form>
</body>
</html>
```

Figure 25: Media Viewer ASPX Page with Service Types.

The consumer website now just provides the required Service Type Id when accessing the media viewer web service methods to identify which service type to use to upload or view the desired media file as shown in the figure:

```
<%
UniversityWebServiceuniws = new UniversityWebService ();
String mediaURL = uniws.GetMediaFileURL (1,"profilepic1.jpeg");
%>
<a target="_blank" href="<%= mediaURL%>">Click Here To View
The Profile Picture</a>
```

Figure 26: using GetMediaFileURL Web Service Method from the website with service types.

This previous sample will get the media URL of the media file profilepic1.jpeg from within the Registration sub system in the media storage.

This enhanced sub system technique can be further improved to allow more coding readability and minimize errors by using the Service Keywords with their IDs in the database as dynamicEnumeration Type values to be used by the service consumers.

An Enumeration type (or Enum type) is a value type defined by a set of named constants of the underlying integral numeric type, this Enum can be generated dynamically to create a Dynamic Link Library file (DLL) containing all the service types from the database service table, and allow the media service consumers to access these service types through there Keyword values instead of the integer ones, which would be easier to read and avoids the error of using a wrong service type IDs.

To create the dynamic service types Enum, a simple code can be used to loop through the services and generate a Key-ID Enum pair for each service in the database as shown in the figure:

```
AppDomain currentDomain = AppDomain.CurrentDomain;
AssemblyName name = new AssemblyName("UniServiceTypes");
AssemblyBuilder assemblyBuilder =
currentDomain.DefineDynamicAssembly(name,

AssemblyBuilderAccess.RunAndSave);
ModuleBuilder moduleBuilder =
assemblyBuilder.DefineDynamicModule(name.Name,name.Name +
".dll");
EnumBuilder myEnum =
moduleBuilder.DefineEnum("EnumeratedTypes.UniServiceTypes
e",TypeAttributes.Public, typeof(int));

foreach (// loop through the SIDs, SKeys from the Database table)
{
    myEnum.DefineLiteral(SKey, SID);
}
myEnum.CreateType();
assemblyBuilder.Save(name.Name + ".dll");
```

Figure 27: Create the dynamic Service Types Enum DLL.

This created DLL file that contains the dynamic Service Types Enum, can be accessed from all the media service consumers, in order to use the Enum as the service type as shown in the following figure:

```
<%
UniversityWebServiceuniws = new UniversityWebService ();
String mediaURL = uniws.GetMediaFileURL
(UniServiceTypes.REG,"profilepic1.jpeg");
%>
<a target=”_blank” href=”<%= mediaURL%>”>Click Here To View
The Profile Picture</a>
…
```

Figure 28: using GetMediaFileURL Web Service Method with service types from the Dynamic Enum.

4.2Providing Consumer Authentication

If the web service is published online, and the service consumers are connecting to the service though the internet, then there is a great risk in hand in case the URL of the web service is known to any unauthenticated users, because in this case the unauthenticated users can use the web service to upload or view files from the Media storage, which should be quite an easy task for a hacker. In this case further security concerns should be taken into consideration, with the main step should be to enforce a password to be sent from the different consumers of the media web service, this password can be used both as a way to authenticate the access to the web service, and also can be used to allow different consumers to access different web methods of the web service if needed.

```
[WebMethod]
public String GetMediaFileURL(String password,int ServiceTypeID,
String FileName)
{
if (//check usagekey is authenticated and can access this web
method from database)
{
String ServiceTypeIDEnc =
System.Web.HttpUtility.UrlEncode(MyEncrypt(ServiceTypeID));


FileName =
System.Web.HttpUtility.UrlEncode(MyEncrypt(FileName));


return
“http://www.myUniversityDomain.com/myUniversityWebServices/M
ediaViewer.aspx?ServiceTypeIDEnc="+
ServiceTypeIDEnc+”&filename="+ FileName;
}
else
{
throw new Exception(InvalidKeyExceptionMsg);
}
return “”;
}
```

Figure 29: GetMediaFileURL Web Service Method with authentication

In this case we are pretty sure that only the consumers who have valid passwords can access the web methods even if the web service is on public networks or on the internet.

5. SHARED MEDIA WEB SERVICE ARCHITECTURE ANALYSIS

The proposed shared media architectural techniques providedevelopers and programmers with great flexibility to manage multiple related websites, projects orapplications in the best possible way. These techniques have offered a lot of benefits but at the same time have had some limitations.

5.1 Advantages of using a Shared Media Web Service Architecture

Some of the advantages of using a shared media web service architectureare:

- After preparing the media web service architecture for the first time, any new website or project would save itself time and effort to add the media management functionality.
- Projects and Websites can be deployed in a load balancer model, to reduce the traffic on one site, without any problems regarding any missing or duplicate media files or consistency issues.

- Archiving systems would be much easier because of the minimum size needed for them, because all the media files are separate from the original projects.
- Server administrators can move the storage location or increase its size or even install new anti-virus soft wares on it without any need to change any application that uses the media web service.
- Any new application with different programming languages or development environments can be easily added to the business with minimum effort and time because all the functionally is already prepared by the web service and all the media files are shared in a cross-platform technique.

5.2 Disadvantages ofusing a Shared Media Web Service Architecture

Here are some of the disadvantages of using a shared media web service architecture:

- It needs more time initially to set up the needed media web service architecture, although it saves greater times later on.
- Some extratime is needed to upload and retrieve the media files from the web service in opposite to using local projects to store the media files.

6. CONCLUSION

Many businesses need to have multiple applications or projects serving the clients and users of it, these applications can vary in programming languages and development environments. Developers of such multi-system environments are forced to deal with the overhead of dealing with shared media files. In this paper, a shared media web service architectural design and implementation is represented and discussed in details to suggest the best possible techniques of such a business architecture to provide the consistency and flexibility among the different business applications and reduce as much obstacles that come with maintainingsuch an architecture.New and improvedideas and techniques need to be researched and invented to provide betterarchitectural designs to try to minimize the few disadvantages that still exist and to fulfill the growing changes and demands in the world of business application developments and programming technologies.

7. REFERENCES

[1] Nawir, F., & Hendrawan, S. A, "The Impact of Website Usability and Mobile Optimization on Customer Satisfaction and Sales Conversion Rates in E-commerce Businesses in Indonesia", The Eastasouth Journal of Information System and Computer Science, 2024.

[2] Vardhan, M., Kushwaha, D.S., "File replication and consistency maintenance mechanism in a trusted distributed environment", CSI Transactions on ICT, Springer, 2013.

[3] Newcomer E., Understanding Web Services – XMLWSDLSOAP, 1st ed. AddisonWesley Professional. Boston, United Stated of America. (2002).

[4] Ugurlu T., Pro ASP.NET Web API: HTTP Web Services in ASP.NET (Expert's Voice in .NET), 1st ed. Apress. New York, United Stated of America. (2013).

[5] "kSOAP2" Kilobyte Objects,[Online].Available: http://kobjects.org/. [Accessed 16March 2025].

[6] Perkins B., Jon D. Reid, Beginning C# and .NET, 2021 Edition. John Wiley &Sons, Inc., Hoboken, New Jersey, United Stated of America. (2021).

[7] Sumartono I., Siahaan A., Arpan A., "Base64 Character Encoding and Decoding Modeling", International Journal of Recent Trends in Engineering & Research, 2016.

[8] Kumar K., Pandey B., Next Generation Mechanisms for Data Encryption, 1st Edition, CRC Press, Florida, United Stated of America. (2025).

[9] Danylko J., ASP.NET 8 Best Practices: Explore techniques, patterns, and practices to develop effective large-scale .NET web apps, 1st ed. Packt Publishing Ltd. Birmingham, UK. (2023).

[10] Millett S., Professional ASP.NET Design Patterns, 1st ed. Wiley Publishing, Inc. Indiana, United Stated of America. (2010).