**Unifying multiple cloud storage services and providing a consistent feature set**

Abhishek Nair

Deparment of Computer Engineering

Shri Govindram Seksaria Institute of Technology and Science

Indore, India

[abhishek\_nair@outlook.com](mailto:abhishek_nair@outlook.com)

Surbhi Shah

Deparment of Computer Engineering

Shri Govindram Seksaria Institute of Technology and Science

Indore, India

[surbhishah81@gmail.com](mailto:surbhishah81@gmail.com)

***Abstract:*** *Cloud storage provides the unique benefit of storing and backing data so that users can access data securely from any computing device with Internet access. With many cloud storage providers in the market, end users have the luxury of having multiple accounts. Managing multiple cloud services and becoming familiar with client application of each provider is inconvenient. Backing up files which may be modified repeatedly hampers the simplicity of data utilization. Security and privacy of data .In light of all these problems, a framework to integrate any number of cloud storage service APIs is proposed. As the proof of capability of the framework, Dropbox and Google Drive have been integrated into the application built upon the framework. The application incorporates a desktop autosync utility that synchronizes user files to the specified cloud storage. Sharing of files across different cloud storage services is also possible. To mitigate security and privacy concerns of the users, the application allows for encryption of files such that encryption key is only available to the users. This application simplifies the life of users by bringing different cloud services under one umbrella using which the users are saved from the trouble of managing multiple services and get features like security, privacy and sharing across the clouds.*

**Keywords:** Cloud Storage, Framework.

1. **Introduction**

Growing requirement for data storage space has led to online storage solutions. Cloud storage [1, 2] is a model of networked storage where data is stored in virtualized pools of storage which are generally hosted by third parties. Hosting companies operate data centres and people who require their data to be hosted buy or lease storage capacity from them. Cloud storage has the characteristics of reliability, agility, scalability and fault tolerance. Presence of multiple cloud storage providers leads to users using multiple storage facilities to store their data. However, accounts with multiple providers make this experience cumbersome.

* Users have to manage multiple login credentials.
* Users are forced to use multiple interfaces to access their files on cloud storages.
* Users cannot share their files with users on different clouds. The other people are also required to use the same service.
* Users find it inconvenient to search their files since the files are on different clouds.
* Users may wish to back up files from a machine to the cloud storage. However, frequent changes to files make it difficult to keep the copy up-to-date.

The inherent nature of cloud storage makes it susceptible to security and privacy breaches. When data is replicated, it is stored at more locations and the risk of unauthorised data access increases dramatically. (For e.g. disposal of old equipment, reuse of drives, reallocation of storage space). Regarding privacy, information experts, computer scientists and entrepreneurs have continued to debate on issues related to data ownership. Users who wish to keep their files private need to encrypt their files.

In this paper, we present a framework which facilitates integration of cloud storage APIs to access multiple cloud storage services along with uniform features such as incremental search, data privacy, auto-syncing, file encryption and sharing across all of them.

The paper elucidates the literature related to technologies that were used in the development of the application. It then describes the framework and the application which has been built upon on the framework. It elaborates the features provided by the application, its architecture and its modules. Finally, it outlines the limitations of the framework.

1. **Literature Review**

Cloud computing has evolved from old technologies. The initial development of this kind of technology developed in the 1950s when large-scale mainframes became prevalent. In the research article by Layton (2010) [1], he discusses the inception of cloud storage due to rising costs of buying and maintaining mainframes. Organisations were not able to provide mainframes for each individual user and so they connected large number of thin clients or dumb terminals, which did not have resources of their own except for an input and output devices, but the users were now able to use mainframes for themselves sharing both storage and resources of the mainframes. As the cost of server hardware slowly came down, more users were able to purchase dedicated servers but quickly ran into different kind of problems related to server management like memory, quality of service, bandwidth, scalability etc. Many small businesses which required server facilities found it difficult and expensive to deploy their own hardware and manage it. Abundant resources available with large organisations and on-demand requirements of large number of users or a group of users sparked the development of cloud services.

The article by Strickland (2010) [2] discusses the inner working of cloud storage. A client (e.g., a computer user subscribing to a cloud storage service) sends copies of files over the Internet to the data server, which then records the information. When the client wishes to retrieve the information, he or she accesses the data server through a Web-based interface. The server then either sends the files back to the client or allows the client to access and manipulate the files on the server itself. Cloud storage systems generally rely on hundreds of data servers because computers occasionally require maintenance or repair. Without redundancy, a cloud storage system couldn't ensure clients that they could access their information at any given time. The article also throws light on two concerns about cloud storage are reliability and security. Clients are not likely to entrust their data to another company without a guarantee that they'll be able to access their information whenever they want and no one else will be able to get at it. Cloud storage providers secure data through encryption and authentication.

## V. I. Levenshtein (1966) [3] developed an algorithm to compute a metric for distance between two string sequences. This is called Levenshtein distance. This metric is calculated based on the number of single character edits, i.e., insertion, deletion and substitution required to convert one string sequence to another. This algorithm has been used to find strings similar to a given string sequence. This allows the system to take into account mistakes made by the user in typing search queries.

## Daemen, Joan; Rijmen, Vincent (2003) [4], in their paper "AES Proposal: Rijndael" give a specification for a symmetric key encryption algorithm called AES (Advanced Encryption Standard). AES uses a user specified key to encrypt the data and decrypt the cipher. There are no known attacks against AES, and thus it is the encryption algorithm of choice for most secure applications. AES has been used to encrypt user data and to ensure privacy and security.

## Fielding, R (2000) [5] in his thesis on HTTP protocol, "Architectural Styles and the Design of Network-based Software Architectures" introduces REST (Representational State Transfer) which is built over HTTP. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. REST has emerged as a predominant web API design model. REST uses already established protocol HTTP to transfer queries and data encapsulated in any data representation format. In this manner REST is scalable, easy to deploy and provides generality of interfaces. The REST architecture has been used to interact with the server and query it for data available with the database.

# Proposed Framework

The framework allows the developers to integrate any number of cloud storage services with the application. It ensures that the interaction between APIs and the application is seamless. It is generalized to accommodate different cloud storage service APIs. This provides an extensible platform to access multiple cloud storage services. New functionalities can also be added to the framework.

In class-based programming, the factory method pattern is a creational pattern which uses factory methods to deal with the problem of creating objects without specifying the exact class of object that will be created. The project utilises this design pattern to create instances of cloud storage service classes so that there object can be created on demand. The class *CloudFactory* is a concrete class which is responsible for creating instances of classes that implement *CloudInterface*. The object requester requests for an object by identifying the class of the object using an identifier. In the framework, we have used the name of the class as the identifier. The names are stored in a configuration file. Any future addition to the cloud implementation will require the developers to write the name of their cloud in this configuration file. The name of the cloud should match the name of the class corresponding to that cloud. To include a new feature in the application, the developers can add a new function declaration in CloudInterface and all the classes that implement the interface need to implement the definition of the new function.

# Use of the framework to develop an application

The application that has been developed incorporates the framework described above. The application can be deployed as a web service providing a user interface which can be accessed via browser. The web application gives user on-the-go access to his cloud storages. The application uses Web APIs provided by cloud storage service providers to access user data stored on them. The Drive SDK [6] gives a group of APIs along with client libraries, language-specific examples, and documentation to develop apps that integrate with Drive. The Dropbox Core API [7] provides access to the files stored over Dropbox accounts. These APIs provide RESTful access to cloud storage services.

AES Encryption [4] has been used to encrypt the files for security and privacy. User files are encrypted using a two level encryption system. The files are encrypted using a randomly generated key and the key is encrypted using user’s passphrase. This allows users to change passphrases at a later stage without the need of re-encryption of files. The application also stores hash to the passphrase so that it can verify passphrase before decrypting files. This saves computational power and network bandwidth.

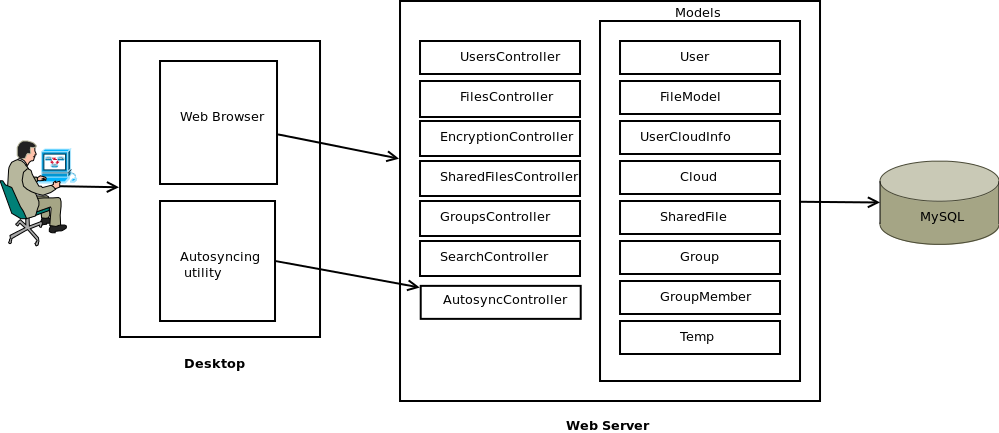
Searching for user files across all cloud storages has been implemented using Levenshtein algorithm. This enables error tolerance over the search strings and takes typing mistakes in consideration. Application provides instant searching giving results as soon as users type in.

The application includes an auto syncing utility that allows users to backup files and folders to user specified cloud storage service and keep data up to date. The desktop utility uses file system Watch Service API to get notifications regarding modifications to files. As soon as the desktop utility is informed of the modification, it propagates the changes to the web application and subsequently to cloud using RESTful calls.

The application supports sharing across different cloud storage services. Sharing is implemented at the application level and not supported by the cloud storage providers.

# Architecture

One of the fundamental design goals of our architecture is to utilise MVC (Model-View-Controller) design pattern for Web application development. MVC pattern enables separation of concerns and allows for clean and loosely coupled code which is easy to develop and maintain.



*Fig.1. System Architecture*

Apart from the Model-View-Controller files the application incorporates helper classes, API classes, routes files and database related files that provide database definitions. Helper classes provide utilities used by other classes that are not part of the definition. API classes provide properties and behaviour required to access the Cloud Storage Service and also perform operations on that service.

1. **Classes**

This module includes all the helper classes and API classes.

* 1. Dropbox

This module contains all Dropbox API library classes for access and storage management.

* 1. GoogleDrive

This module contains all Google Drive API library classes for access and storage management.

* 1. lib

lib module contains all helper classes and interface definition for the application.

1. CloudInterface

This provides the framework that will help integrate different services to the projects interface.

1. CloudFactory

It returns an instance of the cloud storage service client based on the name identifier of the cloud service. All classes whose instances are returned must implement the CloudInterface.

1. Encryption

Encryption class handles encryption and decryption of string input and random key generation. This module uses AES-256 to encrypt and decrypt data.

1. Dropbox

Implements CloudInterface and overrides all the function definitions to provide access to Dropbox API. Instantiation of the objects of this class is done by cloud factory.

1. GoogleDrive

Implements CloudInterface and overrides all the function definitions to provide access to Google Drive API. Instantiation of the objects of this class is done by cloud factory.

1. Utility

Utility class encapsulates different functions that are used in the application like function to create zip files, functions for path handling, function for JSON encoding etc.

1. **Models**

All models are responsible for interaction with the databases and handling core logic for the application. Models abstract all the queries that are maybe required by the controllers. The query definition and execution is further abstracted by *Eloquent Object-Relational Mapping* (ORM) allowing the application to connect to different kinds of databases without changing the definitions of tables and queries. Eloquent ORM is a part of Laravel [8] Framework.

1. User

This model handles the DB interaction for the *Users* tables which contains the data for each registered user. It also defines the constraints the users table follows in the database.

1. UserCloudInfo

This model handles the DB interaction for UserCloudInfo which contains data related clouds registered by the user. It contains the access token retrieved from the cloud storage service that allows access to that service.

1. FileModel

It handles the DB interaction for user files and along with file related metadata, like file ID, file path, boolean value to know if file is a directory and timestamp. Metadata also includes information provided from the cloud storage services.

1. Temp

Handles the DB interaction for temporary files stored at the server which act as cache for frequent file download or in case of a network error.

1. Cloud

Handles the DB interaction for Cloud metadata and cloud related data like cloud ID and API keys.

1. SharedFile

Handles the DB interaction for files shared among users and related metadata. It provides methods to access files shared by the user and files shared with the user.

1. Group

This model handles the DB interaction with groups table. Group refers to share groups to which user can share files to. This model provides methods that create, search and delete group details.

1. GroupMember

Handles the DB interaction with group\_member table and contains data about the users in a groups and metadata related to each user.

1. **Controllers**

Controllers handle the interface between models and views.

1. UsersController

It is responsible for user authentication, registration to application and forwarding user to the dashboard which is the main user interface of the application allowing user to perform all the integrated tasks with the cloud storage services. It delegates appropriate method corresponding to the route calls pertaining to users. It also interacts with the users table.

1. FilesController

Handles route calls pertaining to files and delegates to appropriate method corresponding to the route calls responding with views and controls the database using the FileModel. It contains methods related to files and folders like create folder, upload files and download file.

1. EncryptionController

Handles route calls pertaining to encryption and decryption of files and delegates to appropriate method corresponding to the route calls responding with views and controls the database using the Encryption model.

1. SharedFilesController

Handles route calls pertaining to sharing of files and delegates to appropriate method corresponding to the route calls responding with views and controls the database using the SharedFile model.

1. GroupsController

Handles route calls pertaining to creation, deletion, update of share groups and delegates to appropriate method corresponding to the route calls responding with views and controls the database using the Group and GroupMember model.

1. SearchController

Handles route calls pertaining to different search operation like file searching, groups and user searching for sharing. It delegates the appropriate method corresponding to the route calls responding with views and controls the database using the SharedFile, User and GroupMember model.

1. AutoSyncController

Handles REST route calls generated from the desktop utility that handles auto-syncing with the service. It delegates the appropriate method corresponding to the route calls responding with views. This controller calls other controllers to serve route calls.

1. **Views**

Views module has all the front end code that is used to render the web user interface accessible to the users. It also handles all the AJAX requests generated by the user interface and results of these AJAX interactions.

1. **Routes**

Routes catch all the REST calls to the application and bind them to appropriate controllers in the backend.

1. **Migrations**

Migration is laravel specific module that provides developers with functionality similar to version control system for database so that we can make changes to database definition with relative ease. The application has migration definitions for all classes.

1. **Limitations**

* **Susceptible to change in APIs**

The application is susceptible to changes in the APIs. The implementation of each cloud which has been integrated into the framework is dependent upon the interface exposed by the respective cloud storage API. Changes to the API can be independently done by the cloud storage provider and those changes will have to be accommodated by making changes to the implementation of that cloud in the application.

* **Compromise on individual features**

To provide consistency across all cloud storages, the application compromises on individual features offered by cloud storages. Features unique to particular cloud storage may not be provided by other services.

1. **Conclusion**

This paper describes a framework that facilitates integration of cloud storage APIs to provide an extensible platform to access multiple cloud storage services. Apart from addressing the specified problems the application incorporates features like sharing across different clouds, auto-syncing and data privacy which enhances user experience and assuages privacy concerns of users. In view of all the requirements, the framework has been designed keeping separation of concerns at focus so as to facilitate modular development. To achieve minimum coupling the project has been built over Laravel MVC Framework. In the current version of the application the project incorporates following functionalities:

* Framework to integrate application with cloud storage service API.
* Integrated Dropbox and Google Drive APIs into the application.
* AES-256 encryption and decryption of files that are uploaded to different cloud storage services.
* Sharing of files stored with different cloud storage services between users and share groups.
* Incremental error-tolerant using Levenshtein edit-space algorithm to search files across all the cloud storage service accounts added by the user.
* Auto-syncing functionality Nautilus-python context menu API and Java Watch Service API which allows user to synchronize files on his personal machine with the cloud storage service user specifies.

To create a framework that facilitates integration, the developers had to compromise on individual features provided by cloud storages. GoogleDrive and Dropbox have been integrated as a proof of the capability of the framework. However, these realizations will break if the API interfaces are changed. The developers expect future realizations of the framework to be done by the cloud storage service providers. The existing functionalities can be enhanced further by including online preview and editing of files, versioning and multi-platform support for autosyncer.

**References**

1. Jeffrey B. Layton. Cloud Storage Concepts and Challenges, http://www.linux-mag.com/cache/7617/1.html, 2010, Accessed in Oct, 2013.
2. Jonathan Strickland. How Cloud Storage Works, http://computer.howstuffworks.com/cloud-computing/cloud-storage3.htm, 2010 Accessed in Oct,2013
3. V. I. Levenshtein(1966). "Binary codes capable of correcting deletions, insertions and reversals."
4. Daemen, Joan; Rijmen, Vincent (2003). "AES Proposal: Rijndael"
5. Fielding, R (2000). "Architectural Styles and the Design of Network-based Software Architectures: Chapter 5, REST"
6. Google Drive API. Retrieved from <https://developers.google.com/drive/> Accessed in Nov, 2013.
7. Dropbox Core API. Retrieved from <https://www.dropbox.com/developers/core/docs>, Accessed in Nov, 2013.
8. Laravel Eloquent ORM: Retrieved from <http://laravel.com/docs/eloquent>, Accessed in Oct. 2013.