Design Specifications

1 Introduction

1.1 Intended Audience

This system is primarily meant for use by mobile game developers using the Gideros environment. This system is meant to be simple and easy to use, so that any developer can quickly and easily implement cloud data storage in their application.

1.2 Purpose

The purpose of this system is to provide a simple and easy to use cloud data storage system to Gideros developers, assuming that the developer has a server to use this system on. This system could be used for backing up user's application data in case of device failure, or if the user wants to move their data to another client application on another device. This system is meant to be highly general, so that it can be used for many purposes up to the developer's discretion.

1.3 Goals

- Being a simple, quick, and easy method of implementing cloud storage in a developer's application.
- High generality to allow for a wide range of use cases.
- Having reasonably durable error handling and security.

1.4 Non-Goals & Limitations

- Not being optimized for optimizability by the developer.
- Not being particularly optimized for speed.
- THIS SYSTEM SHOULD NOT BE USED TO STORE SENSITIVE DATA, it is missing security necessary for such a use case.
- CloudBackup does not supply a server. The developer must already have a functioning server.
- CloudBackup is not made for general Lua, as it is dependent on Gideros libraries, however it can be easily ported for general Lua use.

2 Overall Structure

- Lua Object Oriented Interface
  - CloudBackup Objects
    - Initialized Properties
      - Server URL
      - User Identification
      - Data Tag
    - Methods
      - Store Data
      - Retrieve Data
      - Get Timestamp of Storage
      - Get All Tags of User
  - Originate Table
- Developer Supplied Server
  - MySQLi Database
    - Single Relation
      - Fields
        - Unique Record Identifier
        - User Identification
        - Data Tag
        - Timestamp of Data Storage
        - Potentially Huge Data
  - PHP Request Processing Scripts
    - Store Data .php
    - Retrieve Data .php
    - Get Timestamp of Storage .php
    - Get All Tags of User .php
    - Database Connection .php
    - Originate Table .php

3 Developer Interface

The system provides a Lua/Gideros module that provides the CloudBackup class. This class serves as the interface to this system for the developer.

```
CloudBackupInstance = CloudBackup.new(server_url, user_id, tag)
```

3.1 Properties

3.1.1 server_url

The url of the server where backup data is stored.

3.1.2 user_id

A string that acts as an identifier for a single instance of a program. Alternatively, can be thought of as an identifier for a user, like an account/password/username for storing/retrieving data on the server.

3.1.3 tag

A name given to a piece of data. Data is stored/retrieved at the level of tag. This interface does not accommodate storage/retrieval of multiple tags of data at once, nor particular portions of data within a tag.

3.1.4 status

CloudBackupInstance.status

Communicates whether the state of the object is valid or not. The object would be invalid if it was initialized with unacceptable values. It can be checked for equality to either CloudBackup.STATUS_OK or CloudBackup.STATUS_INVALID_STATE.

3.1.5 status_message

CloudBackupInstance.status_message

Communicates details on how the status is invalid.

3.2 Methods

The methods of CloudBackup take parameters on_complete_function and on_fail_function. The methods use the three properties of the initialized CloudBackup to function. They create an event and listener, so that the program is not paused during the http request and server-side actions. Upon reply, if no connection then run on_fail_function, otherwise run on_complete_function. on_fail_function has no parameters.

CloudBackupInstance:method(on_complete_function, on_fail_function)

3.2.1 CloudBackup:storeData

Used to backup a tag of data to the server. Takes an additional parameter: data. Passes to on_complete_function: the timestamp of storage on the server, and an error message communicating full success or the type of error that occured.

CloudBackupInstance:storeData(on_complete_function, on_fail_function, data)

on_complete_function(backup_date, error_message)

3.2.2 CloudBackup:retrieveData

Used to retrieve a tag of data from the server. Passes to on_complete_function: the timestamp of storage on the server, an error message communicating full success or the type of error that occured, and the retrieved data.

on_complete_function(backup_date, error_message, data)

3.2.3 CloudBackup:getBackupDate

Used to get the timestamp of storage on the server for a tag. Passes to on_complete_function: the timestamp of storage on the server, and an error message communicating full success or the type of error that occured.

on_complete_function(backup_date, error_message)

3.2.4 CloudBackup:getTags

Used to get a table of every tag that has been stored by the user_id. This method does not get any data, only their tags. Passes to on_complete_function: an error message communicating full success or the type of error that occured, and an indexed table of tags.

on_complete_function(error_message, tags)

3.3 Setup

A collection of files will be provided for the developer to place in their server. Once in place, the following actions must occur in order.

3.3.1 Database Creation

A MySQLi database must be created along with credentials to access this database. This sensitive information must be placed within the Database Connection PHP file and saved.

3.3.2 Table Creation

The table within the database must be created by running the originateTable function which comes with the module alongside the CloudBackup class. It requires one parameter: server_url. After running, it is then recommended to remove the Originate Table PHP file from the server.

originateTable(server_url)

4 Database Schema

```
primary_key   |   UNIQUE AUTO_INCREMENT
user_id       |   STRING
tag           |   STRING
backup_date   |   DOUBLE
data          |   LONG_STRING
```

5 Security

5.1 Measures Taken

5.1.1 SQL Injection

SQL injection is prevented through use of parameterized queries.

5.1.2 XSS

Tags and data may not contain the following characters: &, ‘, “, <, >. Any tag or data containing any of these characters will raise an error message, and not be stored.

5.1.3 MySQL Login Info Security

MySQLi login security is irrelevant due to the host being ‘localhost’.

5.2 Known Security Flaws

5.2.1 No HTTPS

HTTP is used instead of HTTPS, so data and user identification are sent unencrypted in URLs.

If such security is of importance to a developer, they can modify CloudBackup to use HTTPS.

5.2.2 Storage Parasitism

Any person can store data using the developer’s instance of the CloudBackup system. If they create URLs with valid structure, they can store data using their own created user_id, as well as mimic all other URL structures used by CloudBackup.

If security against this is of importance to a developer, they can modify CloudBackup server-side to include a user_id validation step. Many methods exist for implementing such a step; here is one. Client-side: the developer could create some password to be appended to the beginning of all user_ids of valid clients. Server-side: user_ids will only be accepted if they begin with this password, and if accepted, the password is removed from the user_id prior to the user_id being used in queries.