A
PROJECT REPORT
ON

Cloud-based File Storage System

SUBMITTED BY

Vineet Maurya

ACADEMIC YEAR 2024-25

S.Y. B.C.A. SEM-3

UNDER THE GUIDANCE OF

Name of Faculty

Navrachna University

SUBMITTED TO

Navrachna University

A
PROJECT REPORT
ON

Cloud-based File Storage System

SUBMITTED BY

Vineet Maurya

ACADEMIC YEAR 2024-25

S.Y. B.C.A. SEM-3

UNDER THE GUIDANCE OF

Name of Faculty

Navrachna University

SUBMITTED TO

Navrachna University

ABSTRACT

This project represents the design and development of *The Files Spot*, a file hosting and sharing service. It is aimed squarely at users who want more powerful functionality than what the existing file upload services provide. It leverages a flexible, web-based architecture to ensure that the service is available on whatever platform the user is on. The backend service is written in PHP and serves a basic HTML/CSS/JS frontend. The main features of this service include a very lightweight and responsive frontend for casual use and a flexible JSON API for those who want more control over their file management workflows.

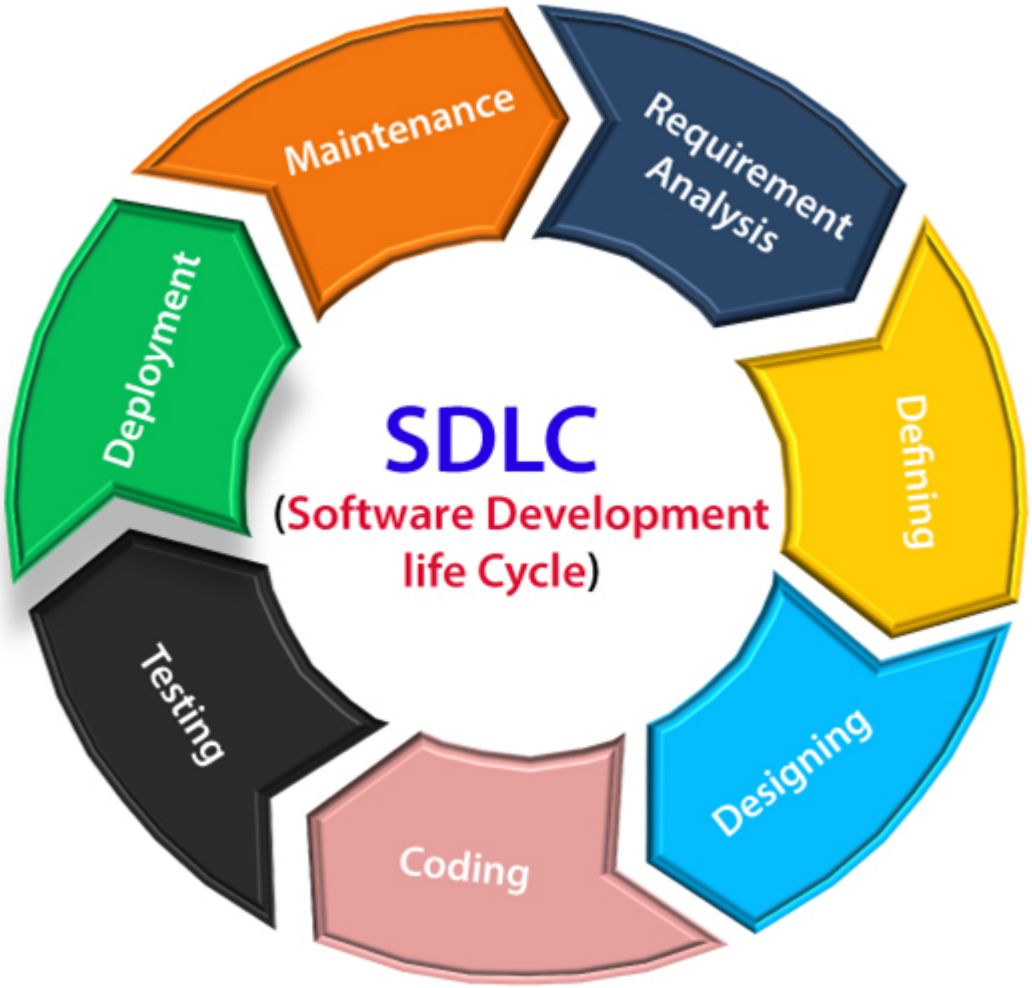
ACKNOWLEDGEMENT

PROJECT PROFILE

Student Information	
Name	**Enrollment Number**
Vineet Maurya	23000068
Project Details	
Project Title	The File Spot - A Cloud based File Storage System
Duration	Insert Duration Here
Name of Project	Insert Project Name Here
Platform	Web
Team Size	1

INDEX

SDLC OVERVIEW



REQUIREMENT GATHERING AND ANALYSIS

2.1 Organization Details

This service was made to fulfill part of the requirements for BCA III semester.

- **Name of Organization** Bachelor of Computer Applications, School of Engineering and Technology, Navarachna University, Vadodara
- **Brief Details of Organization** The School of Engineering and Technology, Navrachna University is a place where students are taught deatils of various technical fields.

2.2 Meetings

2.3 Data which will be Input into the System

The service, by design, aims to require minimal data from the user. Aside from the username/password combo, only files themselves are the only other major form of data that most users are expected to input into the system.

Listed here, in no particular order, are all of the different kinds of data that can be input into the system:

- **Account Details**

  Account Details include data like usernames (arbritary string data), passwords (arbritary secure string data), and recovery email addresses (personal string data). The usernames are unique per-user and will be stored into the application database as-is. The passwords will be salted, hashed, and stored in a database and correlated with the username. The emails will be stored in a database as-is, correlated with the username. Note that emails are optional, and thus a username may/may not have an email address associated with it.

- **Files**

  Files include the files themselves as well as any metadata that is associated with or generated from them. The metadata itself is associated with a file and is stored within the application database, and includes data like `upload_date`, `uploader_account`,

etc. The files themselves are stored in a seperate storage space. This space is accessed only during file upload/download processes. All other interactions refer to the file's metadata stored within the application database.

- **Connection Metadata**

  Some **Connection Metadata** about every file access is stored in the application database. This is mainly intended to control file access and to provide that data that powers the user's file access dashboard. For all users, this data includes access time. For logged in users, the data additionally includes the user's identity. For logged out users, this includes the connecting IP address. In both cases, simply visiting/accessing a page/file *does not* set any kind of cookies or any other persistent storage on the visitor's user agent. Therefore, there is no data to see if the same user accessed the same page/file multiple times.

2.4 Data which will be Output from the System

Most of the data that will be output from the system will be the files that were previously uploaded by users. The system does not generate any user-facing files by itself.

The system does generate logs. These logs will be stored to a file on the server that is running the service. These logs will be accessible only from the server admin side.

2.5 Type of Project

This is a web-based service with seperate front and back ends. The back end is written in PHP with a MySQL database for data storage. The front end is a HTML/CSS/JS frontend that runs on the client's browser. Both sides communicate with each other over a RESTful JSON API.

2.6 Method of collecting Data

Data about other similar services was collected by using the services themselves and by reading about the experiences other people had when using the service. Particular attention was paid to the pain points that surfaced during use and from other users online.

SYSTEM REQUIREMENT SPECIFICATIONS

3.1 Introduction

The Files Spot (henceforth reffered to as TFS) is a simple file-storage service. It allows for simple and easy file backup, organization, and sharing.

3.1.1 Purpose

Many cloud storage solutions are currently available for general use for the public. These are usually either barebones storage services like Amazon S3 aimed at developers to build on top of, or customer-friendly services like Google Drive and Microsoft Onedrive. Unfortunately, services between these two extremes are few and far between. The Files Spot aims to fill this gap in the cloud storage space.

TFS is aimed squarely at power users who want a simple cloud storage service to store and backup their files in. Towards this end, TFS aims to deliver on two fronts - a simple user interface that makes it easy to get started, and a flexible API that allows for easy automation and integration with existing automation tools.

3.1.2 Document Conventions

The File Spot is henceforth reffered to as TFS in the rest of this document.

Users in this document is used to refer to anyone who uses the service, either via the provided web client or via the API.

In this document, clients refer to both the web client and to any service that consumes the service via the provied API. This includes, but is not limited to, various automation services that may use the service by scraping/parsing the web client instead of using the API.

All code in this document is `written like this`. It will be case-sensitive, and is intended to be parsed as-is.

Some portions of this document will require the user to replace the placeholder values with their own values. The placeholders will be formatted as `<placeholder>`. The user/reader is expected to replace the entire placeholder text (including the < and >) with the appropriate value of their own.

Some other portion of the document may require the user to put the result of some operation on some data and replace the placeholder with the result. These operations are formatted *like this(<with the data indicated as placeholders>)*.

3.1.3 Intended Audience and Reading Suggestions

The intended audience for this documernt includes all stakeholders and any user who wishes to know more about the workings and design principles of the service. This should be especially useful for users who are wanting to know about the rationale behind certain decisions made during the development process.

3.1.4 Project Scope

The project explicitly aims to tackle the problem of uploading and downloading files. As such, support any and all file formats is within hthe scope of this project.

Security is provided via a basic login/token wall. The communication may be encrypted using standard HTTPS/TLS protocols, discussion about said protocols is out of the scope of this document.

File and data/metadata storage is a core part of this service and will be the main point of discussion in this document. Strategies to manage file access, data storage, and verification of file integrity will be an integral part of this project.

Encryption of files at rest is an explicit non-goal of this service. This is to avoid potential legal complexities regarding hosting of potentially illegal content. As such, all files uploaded to this service will be visible to the admins of the service. This is ordinarily meant to allow admins to comply with legal demands, but users should make sure that they don't upload any sensitive data to this service.

3.2 Overall Description

On the user side, The Files Spot (TFS) is a simple web application that can be accessed from any web browser. It will have a UI for uploading new files to the cloud and a UI to manage files that have already been uploaded to the cloud. The UI will be very simple and minimalistic to ensure that it can be easily parsed by tools and other accessiblity tools.

On the backend, it will be a simple file storage service written in PHP that will manage files for multiple users. It will also allow users to share files - both to specific users or to the public at large. File management will be primarily accomplished via tags set by the user.

3.2.1 Product Perspective

3.2.2 Product Features

The web client is designed to be easy to use for all kinds of people on all kinds of connections. Specific care is taken to ensure that the design is easy to understand and interpret.

The underlying DOM of the web client is designed to be minimal and clean to ensure that various accessiblity services can easily surface relevant information to the user. Although this is a non-goal, an attempt will be made on a best-effort basis to ensure that the DOM itself remains relativey stable so that any scrapers that rely on it can remain functional as long as possible.

The API is meant to be very easy and intuitive to use. This is accomplished via the use of standard HTTP verbs (GET, POST, PUT, PATCH, DELETE) in the various file descriptors. The API will also be JSON-based and stateless (as far as possible) so that it is easy to inetgarte into existing CLI-based and app-based workflows.

3.2.3 Use Cases and Characteristics

The primary usecase for this service is to easily backup files to the cloud. In addition, the simple protocols make this a very accessible application.

Another use case for this service could be file transfer and sharing. It explicitly supports sharing with various people (or the world) to enable this usecase.

3.2.4 Operating Environment

This service can be run on any service that can run PHP.

People looking to deploy this service should first check what version of PHP their host supports. If a supported version is available, refer to the PHP deployment guides for that host.

Developers wanting to fork/make changes should refer to the comments in the source code. Additionally, we reccomend that they install just as a command runner to simplify their lives. This project makes heavy use of just recipies to automate mundate CLI tasks. For reference, this service was originally built and tested on PHP 8.3.10 running on Windows 10.

3.2.5 Design and Implementation Constratints

A major implementation constraint here is the need to make this service as user firendly as possible.

The web client aims to follow the principles of progressive enhancement. As a part of it, we ensure that the web app is useable in all three loading stages (HTML, HTML+CSS, HTML+CSS+Javascript). This therefore rules out JS-only frameworks like React, Vue, etc.

Another design decision to not use any large external libraries. This means that we mostly avoid large CSS libraries like Bootstrap. Instead, we opt for custom, minimal CSS and JS to power the site.

On the server side, the decision to avoid JS frameworks means that we need a language which can stringly augment the existing HTML instead of generating a new one everytime one is requested. This leaves us with two major, battle-tested options - Python (via Jinja2 templates in Django/Flask), or PHP. Here, we opt for PHP as it is a lighter, much simpler language to make a web server in. Additionally, PHP comes with a lot of server utilities built in (database methods, etc) which removes the need for many external dependencies.

3.2.6 User Documentation

3.2.7 Assumption and Dependencies

On the server side, the server requires the presence of a PHP runtime on the system. Additionally, the server requires access to a MySQL instance (either locally or on a seperate server) to serve as the application database.

The web client assumes that the user has a reasonably modern web browser and a HTTP(S) connection to the server that's running the backend service. It must me capable of making and parsing various kinds of HTTP requests (GET, POST, PUT, PATCH, DELETE) Additionally, the browser must be capable of parsing and displaying HTML/CSS, with JS required for additional functionality.

The API service simply requires a client that is capable of making various kinds of HTTP requests (GET,POST,PUT,PATCH,DELETE) and handling a JSON response.

3.3 System Features

3.3.1 API Access

The primary feature of this service that differentiates it from other similar services is the presence of a simple, but powerful API. This allows a user to integarte this service easily with their existing setups via something like a bash script that runs on a regular basis.

The API is exposed at the `/api/` endpoint. Access to all but public files require the client to perform some form of authentication.

The easiest way is for the user to generate a user token from another client that the user is already authenticated on. The token should then be passed with every request

to identify the user. The token can be included either as `Authorization: Bearer <user-token>` in the request header, or as `?access_token=<token>` alongside other URL query parameters.

Another way is to use HTTP Basic Auth. In this case, the username and password for the user are directly passed as a part of the request headers. Note that this is heavily discouraged, as the credentials can then be revealed if the connection is insecure or if the user is a victim of a MitM attack. To use this auth scheme, include `Authorization: Basic` *base64(*`<username>:<password>`*)* in the request headers.

Each endpoint supports some subset of the standard HTTP methods. The information for each endpoint is defined as follows:

- `/api/<path/to/file>`
  - `GET`

    Returns file contents with HTTP status `200 OK` if the file exists, and HTTP status `404 Not Found` if the file does not exist. If the user is not authorised to access the file, an HTTP `401 Unauthorized` is returned instead.

    By default, the response includes a `Content-Disposition: inline` header, indicating that the response is intended for display only. Adding `?download=true` causes the reponse header to include `Content-Disposition: attatchment; filename="`*`<filename.ext>`*`"` instead, indicating that the file is intended to be downloaded to the client's system. *filename.ext* is the filename that the file was originally uploaded with.

    By default, the entire file's contents are returned. If the request contains the `Range: bytes=`*`<start-offset>-<end-offset>`* header, the contents of the file within the range `<start-offset>-<end-offset>` are returned with an HTTP `206` status code instead. If the file does not exist, a HTTP `404` status code is returned instead. `<start-offset>` is clamped to the file size, with a default value of 0. `<end-offset>` is also clamped to the file size, with a default value of `file-size`. If `<start-offset>` ¿ `<end-offset>`, an error with HTTP status `400 Bad Request` is returned.
  - `PATCH`

    Updates a part of the file. The part of the file to be updated is indicated by the value of the `Content-Range` header (eg. `Content-Range: bytes`).

    The request **must** contain the `Range: bytes=`*`<start-offset>-<end-offset>`* header. The contents of the file within the range `<start-offset>-<end-offset>` are updated with the new contents. If the file does not exist, a HTTP `404` status code is returned instead. `<start-offset>` is clamped to the file size, with a default value of 0. `<end-offset>` is also clamped to the file size, with

a default value of `file-size`. If `<start-offset> <end-offset>`, an error with HTTP status `400 Bad Request` is returned.

- `DELETE`

  Deletes the file.

  If the file does not exist, returns a HTTP `404 Not Found` response. If the user does not have the requisite permissions, `401 Unauthorized` is returned instead.

  Note that a user cannot delete part of a file. Therefore, any `Range: *` headers set on the request are ignored.

3.4 External Interface Requirements

The system requires unteraction with two major external systems: a database server, and a file server.

The database server is assumed to be wither MySQL, or another MySQL-compatible database server like MariaDB. The main app server does not make any other assumptions about the database server. Therefore, complications like database sharding and scaling are the responsiblity of the database server itself.

The service also assumes access to a file server. By default, the files are stored on the same server as the app service itself i.e. the app server also acts as the default file server. The service itself makes no assumptions about the file server itself. Therefore, any file server complications (file duplication, seperate file storage service) will require the developer to write their own adapter for the server itself by `require`-ing and overriding the appropriate class. The default file server included in this app stores the files in a seperate folder on the app server itself.

3.4.1 User Interfaces

The main user interface for this service is its default web UI.

The web UI is made with HTML, CSS, & JS. It aims to be simple, fast, and effective. It allows a user to login, generate/manage user tokens for use with the API, and view/manage files that they have uploaded to the service through various means.

3.4.2 Hardware Interfaces

This app does not make use of any external hardware except those that power the server and the client.

All interaction with the hardware server happens through PHP. Therefore, the hardware is assumed to be one that is capable of running a PHP server.

All interaction with the client's hardware happens through the user-agent that the user uses. This hardware is mostly used to make requests, and store temporary data to ease the load on the app server.

3.4.3 Software Interfaces

The main software interface for this service is its API.

The API is a major part of this service's offering. It is a simple, RESTful JSON API with support for standard HTTP verbs like `GET`, `POST`, `PATCH`, etc.

For a complete documentation, refer to section 3.3.1 on page 7.

3.4.4 Communication Interfaces

The app service is defined as an app-server model with a clear boundary between the server and the client. Communication between the server and the client happens over a HTTP connection using a JSON API as described in section 3.3.1 on page 7.

3.5 Other Non-Functional Requirements

3.5.1 Performance Requirements

The service can handle multiple users uploading and downlaoding files at the same time. Although the exact number of simultaneous users depends on the exact configuration of the server hardware that powers this service, note that there is no restriction within the service framework request for limiting the number of concurrent users.

3.5.2 Safety Requirements

The app service makes no attempt to limit the kinds of content that a user can upload. Therefore, it is the user's responsiblity to ensure that the content that they upload to or access from the service is safe.

3.5.3 Security Requirements

The service only guarantees access-level security i.e. it guarantees that an unintended user (except the server admin) will not be able to access any of he user's files and view/modify them.

The file data itself is also not encrypted both in transit (except using HSTS if configured) and at rest on the server (except when configured by the admin - see 3.4 for a more detailled discussion.) by default. Although the service makes an attempt to ensure that the files stored are returned as uploaded (and warns the user if a difference is found),

note that is could be trivially bypassed by a dedicated attacker. The suer must therefore employ other file integrity checks of their own (preferably out-of-band) to verify that the file downloaded has the intended contents.

3.5.4 Software Quality Attributes

Software quality is upheld by employing a series of tests, both automated and manual. These tests aim to test every aspect of the system and ensure that it matches the expectatons defined in this document.

3.6 Other Requirements

SYSTEM ANALYSIS AND MODELLING

4.1 Use case Diagram

4.2 Normalization and E-R Diagram

Field	Datatype	Constraints and Other Notes
Id	Integer	Primary key, Autoincrement
Name	String	Not null, Unique
Password	String	Non null, Secure
Email	String	Validated
Payment Id	String	From payment provider
Order Id	String	From payemnt provider
Payment Signature	String	From payment provider
Payment Status	String (Success—Failure)	Verified seperately
Payment Verfied	Boolean	Authenticity of payment

Table 4.1: User Table

Field	Datatype	Constraints and Other Notes
Id	Integer	Primary key, Autoincrement
Key	String (UUID)	Unique, Non null
Label	String	Human-readable, User defined
User Id	Integer	References **User** table as defined in Table 4.1

Table 4.2: Token Table

Field	Datatype	Constraints and Other Notes
Id	Integer	Primary key, Autoincrement
Name	String	Unique, Non null, As stored on disk
User Id	Integer	References **User** table as defined in Table 4.1
Last Upload	Timestamp	Non-null
Original Name	String	As uploaded by user
Size	Integer	Non null, Used to calculate storage consumption for user

Table 4.3: File Table

Field	Datatype	Constraints and Other Notes
Id	Integer	Primary key, Autoincrement
Label	String	Human-readable, User defined
User Id	Integer	References **User** table as defined in Table 4.1

Table 4.4: Tag Table

Field	Datatype	Constraints and Other Notes
File Id	Integer	References **File** table as defined in Table 4.3
Tag Id	Integer	References **Tag** table as defined in Table 4.4

Table 4.5: File Tag Table

4.3 Data Dictionary

4.4 Functional and Behavioural Modelling

4.5 Gantt Chart

Field	Datatype	Constraints and Other Notes
Id	Integer	Primary key, Autoincrement
Name	String	Human-readable
Description	String (Long)	Description of subscription benefits
Price	Integer	Positive non-zero integer, defined in the smallest increment (eg. cer
Subscription Id	String	Order item in payment processor
Currency	String	ISO code, Default: INR

Table 4.6: Subscription Table

TEST CASES

SCREENSHOTS

LIMITATIONS AND FUTURE ENHANCEMENTS

CONCLUSION

REFERENCES AND BIBILOGRAPHY