CS5617 Software Engineering

Spec Document - Cloud Upload Module

Yagnesh Katragadda, Cloud Team Member

Team

- B Sai Subrahmanyam (Team Leader)
- Polisetty Vamsi (Co-Team Member)

Overview & Motivation

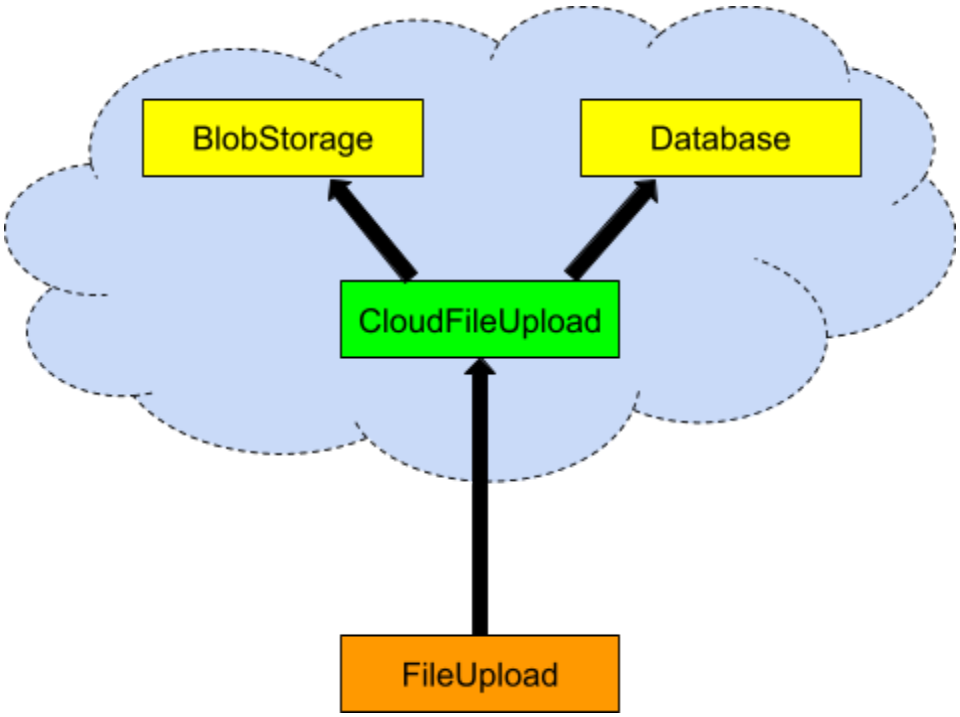
It has been quite common with the emergence of the online classes that the students and the instructors need to hold all labs and submit everything online. However, in an offline setup as well, the lab submissions over a local system are often tedious, not efficient and prone to data loss. In order to solve this problem we leverage the benefits of cloud computing, we intend to develop a cloud service for the Lab Monitoring Session. The cloud module provides a one-stop storage location where the submissions made during particular lab sessions are stored. This helps the client to submit their files and at the same time for the server(teacher) to store the submissions made by the students without the hurdle of local storage. The lab monitor will be able to download all the lab submissions from as well as the students to submit their lab submissions flexibly and at ease. It also ensures that the data that the student upload is secure and can be accessed at a later point in time both from the instructor's end and the student's end.

Features

We would like to provide the following features from the cloud module of the project.

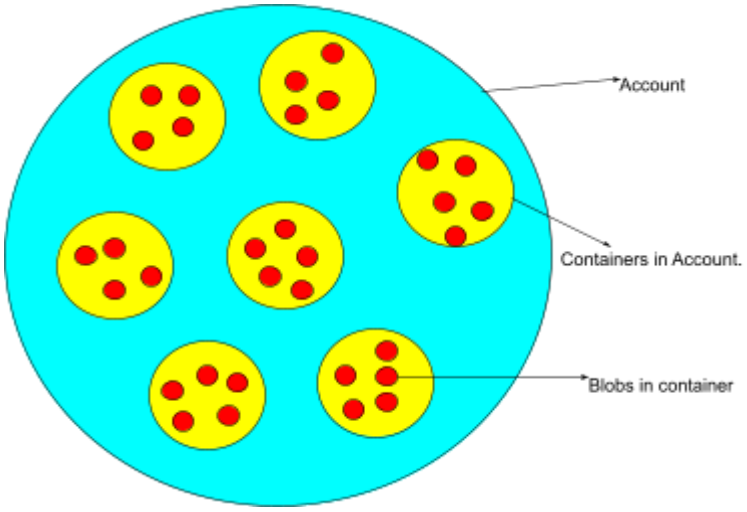
1. To be able to accept answer pdf file's from each of the attendees to the meeting and store them in the cloud.
2. To be able to show all the submission files for all the sessions conducted by a host.
3. To be able to show the attendants all the files he/she submitted in the previous sessions.
4. A simple UX to help the users to perform all the above tasks.

UML - Class Diagram



Design

- **Blob Storage**
  - We will be using Azure Blob Storage for implementing the storage class.
  - It is an optimized cloud storage service provided by Microsoft, especially for the storage of unstructured data like pdf files which is the major requirement for our case.
  - It also extends the flexibility for other file types like images, audio, video etc…
  - All the access to the storage unit is done with the help of the Rest API requests like HTTP and HTTPS requests.
  - The access to the Blob storage objects can be done via HTTP/HTTPS requests like Azure Rest APIs etc….
  - The implementation overview of the Blob Storage is as follows:



- A storage **Account** provides a unique namespace in Azure for your data. We can access the whole storage account using the URL, https://plexshare.blob.core.windows.net/
- A storage account can include an unlimited number of **containers** which can be used one per session. It contains an infinite number of blobs. We

can access the container using the URL, https://plexshare.blob.core.windows.net/<session_id>, where the <session_id> can be replaced with the ID of the session that we would like to access the submissions for.

- A **Blob** is the last level in the hierarchy of the storage, which are the files that we stored in the cloud. A container can store an unlimited number of blobs. We can access each blob with the URL, https://plexshare.blob.core.windows.net/<session_id>/<file_name>, where <file_name> is the name of the file we want and the <session_id> is the id of the session to which the file belongs to.

- **Database**
  - We intend to use the Azure Database for PostgreSQL in order to implement the database for storing the metadata.
  - The reason for choosing this would be due to its ability to handle heavy loads with decent performance, better scalability and the security in terms of illegal accesses and the threats.
  - Since the session data here is very important for future references, it is also vital to have better backup and the same is provided by this database along with other database management functions such as upgrading, patching, and monitoring without user involvement.
  - The relational schema tables that we intend to use for the implementation of the database is as follows:

i) Session Table

ID	Session ID	Username	Start TimeStamp	End Timestamp
Integer	UUID	Varchar	Time	Time

The Session Table maintains the basic details of the session. The session ID is the unique identifier for each session and the respective time periods are stored in the start and the end time stamps. Host Username corresponds to the person who acts as the server for the session. For each new session, a new entry gets added to the session table.

ii) Submission Table

ID	Session ID	Username	Submission TimeStamp	Blob link
Integer	UUID	Varchar	Time	Text

Session Table is another relational table that is implemented in the database which maintains the metadata of all the submissions that have been made by the clients(students). The data that is stored in this case are the user who is submitting the data, the session in which the student has made the corresponding submission, Blob link providing the link to the submission file, time at which the user has made the submission and the unique ID to distinguish the submissions made by the users in a particular session.

CLASSES

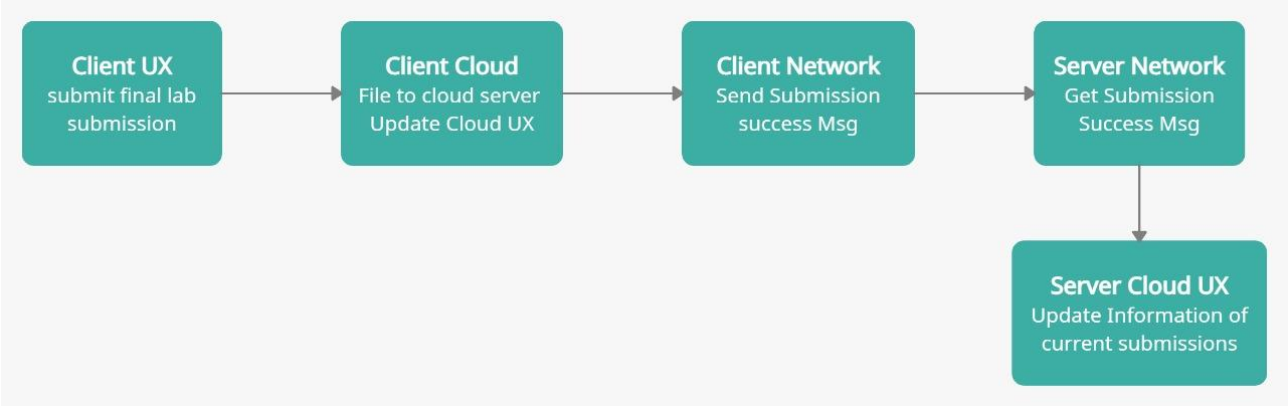
- **CloudFileUpload**
    - CloudFileUpload class is implemented in the cloud which gets requests from the lower class File Upload in order to add the newly submitted file to the cloud.
    - Cloud File Upload verifies the identity of the user and if the user is found to be valid one, it accepts the submission from the respective user.
    - It uses the Blob Storage class's insert file function inorder to insert the file to the container.
    - It uses the database to populate it with metadata of the submission and updates the blob link using the PostgreSQL procedures.

- **FileUploadClass**
    - File Upload class uses the Cloud file upload class that is present in Azure and sends a request to store the files in the cloud.
    - This class is present in the local application unlike the cloud file upload class that is present in the cloud.
    - It provides the functionalities to submit the file.
    - On creation of every new session, this class sends a request to the Cloud file upload class to create a new container to store the files of that session.

- **BlobStorage**
    - This class provides the provision to access or edit the Azure Blob Storage.
    - The insert function of this class inserts a new file by taking the container name(session ID) and the file as the inputs and stores it in the blob.
    - Create container function creates a new container in the blob storage which is called by the cloudfile upload class upon the creation of a new session.

- **Database**
    - This class provides the mechanism to manipulate the data present in the Azure Database holding the metadata.
    - **InsertSession**() function is provided by this class to store the metadata of the session which is invoked by the cloudfile upload class.
    - Similarly, **InsertSubmission**() function is provided to add the submission data.
    - **UpdateSubmission**() function is invoked when an existing row(user submitted a file) is to be replaced(user modifying the submission made).

UML - Activity Diagram



The activity flow for the Upload Module is as follows:

★ First, the client submits his/her assignment for the lab session through the submit option provided by the UX Module.

★ The client then receives a message stating the successful submission of the file.

★ The server network will receive the successful file submission from the respective client. Here, when a packet is sent to the network module, the network module through its publisher-subscriber design pattern, sends a notification to the server machine that a submission has been received.

★ The file gets updated into the Cloud against the corresponding user maintaining it in proper consistent state.

Design Pattern

- **Single Design Pattern →** The cloud module runs on a single design pattern completely as we will be using a single cloud for each session and the database is also the shared one which we use across sessions to store the submission links. It is not a wiser choice to go for multiple cloud objects for each session since the Blob Storage by default has the flexibility to provide proper distinction as discussed above.

Analysis

Blob Storage has support to handle almost 500 requests per second for a single blob which is self sufficient for our case. Also, we have chosen direct client cloud interaction instead of contacting cloud via the server as it reduces the latency for accessing the cloud. This way of interacting with the cloud makes the cloud responsible for managing various requests and is pre-implemented by Azure efficiently. In terms of the security, Azure automatically encrypts the data when it is persisted to the cloud.