The popularity of cloud technologies is undeniable, as more and more individuals and organizations benefit from the scalable, secure and cost-efficient access to computing resources which the cloud provides in a matter of minutes over the internet. The field of cloud computing is still growing, as the global cloud was valued at almost $500 billion dollars in 2022, with an estimated future growth rate of 14%. Many experts also claim that, in the future, 95% of workloads will be carried out in the cloud. Many organizations nowadays migrate their IT infrastructure to the cloud, relieving them of the need to set up and maintain on-site datacenters. High-performance computer clusters and supercomputers can also be rapidly provisioned to perform complex calculations, like weather predictions, and many different applications are accessible via the cloud, such as outlook and zoom, which were extensively used during the pandemic where people were primarily studying and working remotely.

Especially popular is the use of the cloud for storage purposes, as people build data lakes to analyze vast amounts of structured and unstructured data, or back up and replicate their most essential data on the cloud. Cloud storage services can also be integrated into applications to fulfill their storage needs, and they facilitate the sharing, synchronization and real-time collaboration of files among many users. Many such storage services exist, each differing in performance, pricing, capabilities or other aspects, meaning that people might want or need to use more than one at a time. However, each service has its own interface for communicating with the storage backend, making it quite tedious to switch from one service to another. Therefore, the purpose of this project is to provide a single interface that allows interaction with many different cloud storage services.

Specifically, the main goal was to implement a python library which provides this common interface. To this end, it was required to study the official software development kits of each selected service and understand how they can be used to perform authorization and file manipulation. For all supported services, the library should facilitate the upload, download and deletion of files. The performance of the library was also evaluated by measuring the execution time for each service across a variety of workloads.

So, for this project, the python programming language was used along with the official Python SDKs of each supported service, which were: Dropbox, Box, Google Drive and Amazon S3. The logging module provided by the Python standard library was also extensively used, both by our library and the SDKs themselves, providing a unified approach for reporting events to the user, based on their severity. For authorization with most of the services, we relied on the widely used open authentication protocol, using access and refresh tokens to gain access to user files and folders stored on the services. Finally, GitHub was used for version control and to monitor the progress of the project.

When interacting with the library, the user is asked to provide four command line arguments: The name of the cloud storage service that should be accessed (Dropbox), the operation that should be performed (e.g., download) and two paths- one pointing to a file or folder stored remotely on the specified service and the other pointing locally, if applicable. Regardless of the storage service that the user wishes to access, they need to provide these four arguments, establishing in this way a common interface.

As shown here in the design model, the user interacts with the library by providing the four arguments outlined in the previous slide. The library then hides the implementation details and complexities of each storage provider, processing the arguments in such a way that they are accepted by the SDKs in order to communicate with the storage backend. For example, while the Dropbox SDK accepts file paths, the SDK offered by Box requires file ids in order to perform operations, requiring as a result a traversal of the file structure based on the provided path to first locate the specified file. The SDKs also differed in terms of capabilities as, for instance, some did not support the download of folders, which needed to be addressed by the library. Furthermore, the library hides the details regarding authorization by storing the access and refresh tokens that are retrieved after using the authorization mechanisms offered by the SDKs. The tokens are stored and used for as long as they are valid, so that the user is only asked to enter credentials when new tokens need to be acquired. As mentioned before, the library also logs events according to their severity, both those generated by the use of the library and the ones generated by the SDKs, depending on their severity.

In order to evaluate the performance of our library, we measured the time required to upload, download and delete files which ranged from 1MB to 1GB in size. All 3 operations were executed 5 times on each file size, for each supported service, calculating and recording the mean time taken. Larger uploads were conducted in chunks of 32MB to improve reliability.

Regarding upload performance, the library performs roughly the same when uploading smaller files to any service. For files greater than 100MB, however, Google Drive’s performance significantly degrades, requiring twice the amount of time compared to the other services. This is likely due to internal API limitations placed on the speed of chunked uploads, as other API users have also reported this in the past, while changing the chunk size seemed to make no discernible difference either. Overall, uploads made to S3 and Dropbox were the fastest, performing equally well throughout.

When downloading files from the supported services, there are no significant performance differences like those observed during uploads. Specifically, the time required download is consistent among all services for files up to 100MB. For larger files, S3 performs the best the other services aren’t too far behind, and the differences could be attributed to the varying distance from storage servers.

For file deletion, performance was almost identical regardless of the service used and consistent for all file sizes, due to deletion being fast by nature. There is a small variation of around half a second between Dropbox, Google Drive and S3, again likely because of latency. Box is consistently slower than all the other services, however, but only by a second at most.

In conclusion, for this project, a python library that provides a single interface for accessing various cloud storage providers was created. Using the library, four storage services are currently accessible with consistent performance regardless of the service invoked, and support for upload, download and deletion of files is provided. The library is also cross-platform compatible, working across Windows, macOS and Linux, and requires minimal user interaction, meaning that the user could be a person as well as another library system etc. Furthermore, because we applied object-oriented programming, capturing each provider in their own class which inherited from an abstract class, the library can easily be expanded with further operations (e.g., renaming or labelling files), along with other storage services.