

# Storage Feature - SRS

## Team Fortran

### Team Members + Roles (Sprint 1)

- **Project Manager - Zoe Liebenberg**
- Business Analyst - Christian Kenan Devraj
- Designer - Olumayowa Shoderu
- UI Engineer - Olumayowa Shoderu
- API Engineers - Muziwandile Ndlovu
- Backend/Service Engineers - Omolemo Mashigo
- Data Engineers - Larisa Botha
- Testers - Simphiwe Ndlovu
- Developer Operations - Lindinkosi Kunene

## General Description of Feature

### **Context of Feature**

The web application will need storage both locally and externally. Locally that would be the session storage and the latter would be server storage which will consist of a variety of the user's uploaded files. This includes the user's Academic Transcripts, CVs and profile photos.

A reliable database system will have to be implemented to ensure consistent data capturing and file storage. FORTRAN decided that Firebase will be the best storage solution due to its security and convenience. GraphQL will be the API used to upload the user's files.

## **Feature Functions**

The storage functions should be able to store different types of data such as JPEG, PDF, TEXT ect. However, the user will only be able to choose from a limited number of file types. While giving users the ability to upload, download, append and remove data files uploaded by the web application.

Additionally the storage feature will be integrated into numerous other features namely the student/company profiles and the company rep features. This in turn will allow students to upload their credentials and academic transcripts on the undergraduate portal and also allow the respective companies to download these files. The UI of our storage feature as well as the backend implementation will be merged in with the above-mentioned features when fully functional.

## **User Characteristics**

The users can be broken down into 3 main subgroups, which consists of students, industry members and Administrator Users. This is mostly because each subgroup will have vastly different objectives when interacting with the University of Pretoria (UP) graduate's portal.

1. Students – These users will generally be University of Pretoria undergraduates and postgraduates. The main objective of these users will be to advertise their profiles and accomplishments. Students will also be interested in uploading their academic records/CVs to this portal for Companies to view.
2. Companies – This user type will be interested in browsing UP graduates on the portal by viewing their profiles and downloading the students' academic records and CVs.

3. Administrators - These users will be granted special privileges in the UP portal. Mainly the user type will be in charge of managing and moderating what the Students and Companies are doing on the website. Admin users will also have the power to remove uploaded files/profile photos and suspend certain accounts.

## Constraints

- PostgreSQL or Firebase software will need to be used for the online data storage. This promotes consistency among the other features who will use the same software.
- Angular will be used to implement the client-side framework and the UI of our feature. This is required as all the UI of the other features will be programmed using Angular and the storage feature must be compatible in this regard.

## Assumptions and Dependencies

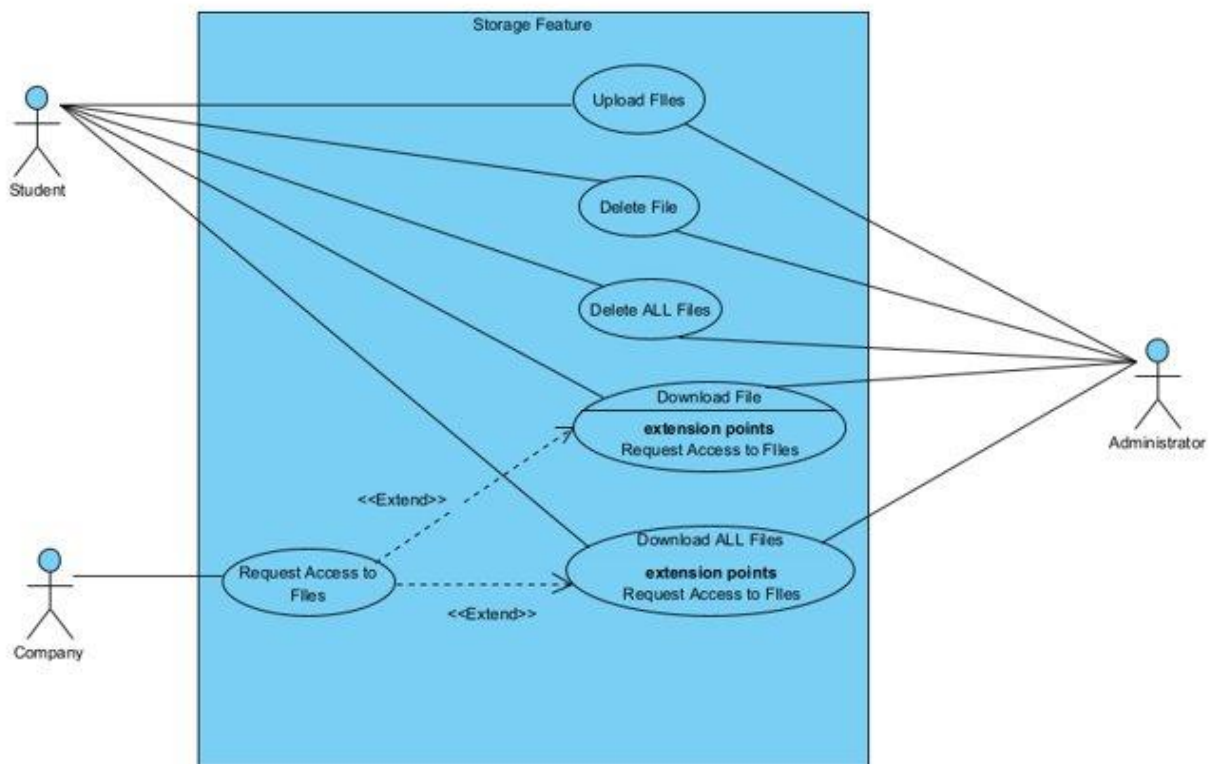
The storage feature will be implemented alongside the user profiles and company features, therefore it is assumed that these respective features will be fully developed for FORTRAN's feature to be functional.

The storage feature's UI will be implemented within the user profiles UI; hence our feature will be fully dependent on the user profile team to provide a suitably implemented system. Although our front-end software will be created independently, a problematic user profile page will not be easy to merge with.

Additionally, it is assumed that the company feature will develop a functional UI to download the students' academic transcripts/CV. We will thus be dependent on the Company Profile team to meet their task or downloading files will not be an available feature for this system.

Lastly the Authentication feature team will need to be utilized as there are many instances where a user (Student or Company) will be requesting access to download certain files from our storage. Therefore the users will need to be systematically verified to maintain security within the mini-project.

## Specific Requirements



## **Functional Requirements**

1. Interface - System must be able to provide a user-friendly interface for the user to upload their files. This includes a "file upload" button as well as a brief menu to upload a file. If the incorrect file was uploaded, the user will be prompted to upload the correct file type.
2. Storage - System must store all the users' files on a database, namely Firebase or PostgreSQL.
3. Data Retrieval - The feature shall allow users who have been granted access to view undergraduate's documents to have access to the files and can download them easily.

## **Non-Functional Requirements**

1. Security - The files uploaded must be stored securely in a database outside the reach of users who are not authorized to view such information.
2. Error Handling - System shall provide simple and understandable feedback when a user makes an error. Such as a pop-up message when the user tries to upload an invalid file type.
3. Usability - System shall upload/download the required files within a reasonable time frame, assuming that the user has a decent internet speed.
4. Reliability - System shall have near perfect reliability when performing tasks with the database, assuming the user has a stable internet connection.

## Preliminary Database Schema

### Upload Table

fileId	userId	filePath	fileCategory	fileExtension
--------	--------	----------	--------------	---------------

### Data Types

- fileId- Integer(10)
- userId- Varchar(255)
- filePath- Varchar(255)
- fileCategory - Varchar(255)
- fileExtension- Varchar(255)

### Descriptions

- File\_ID - The primary key of the table, this ID will be generated dynamically and be used to differentiate and locate the user's respective files.
- User\_ID - The user's unique ID will be stored (may that the Student or Company) as to have a form of reference to identify which user is uploading documents.
- File\_Extension- The type of data file uploaded must be specified, this will assist in differentiating between what the file is being used for. For example, a photo of the student and the student's academic transcript will have different file types.
- File\_Path - This field will store the path of the file, the primary function of this is for data retrieval by the companies that will be useful later on.
- Description - A short description of the file uploaded by the user will be tabulated, however this description may be automated with prepared explanations of the

uploaded file. These descriptions may help the developers differentiate between the purpose of the files

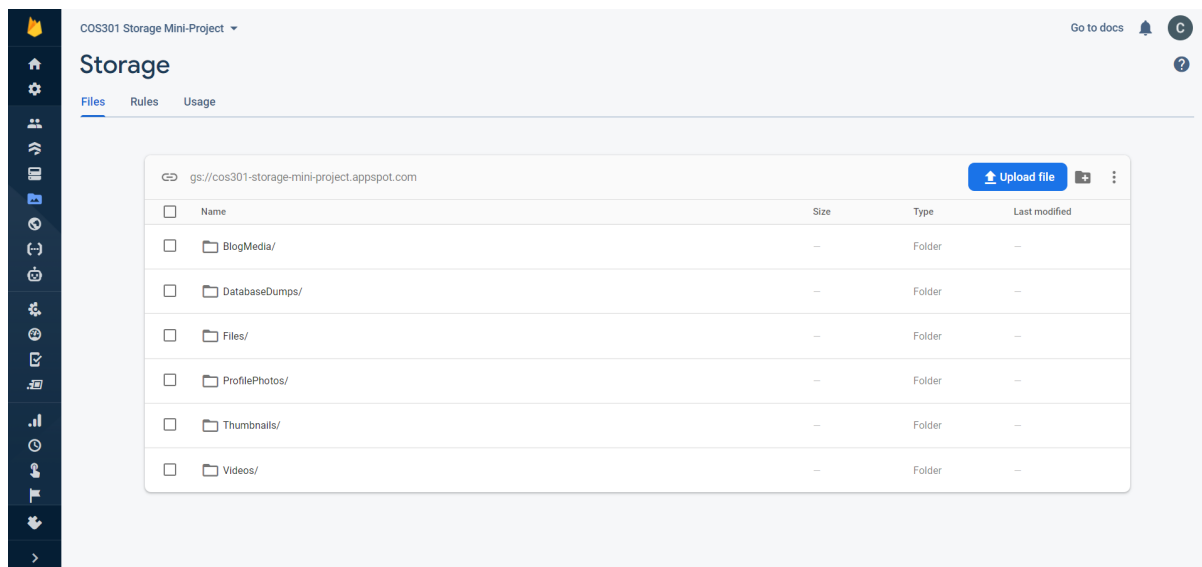
## Database Security Measures

In our efforts to provide a protected and stable storage feature for our users (students and companies) to interact with, Firebase Security Rules was chosen as the optimal solution. Firebase Security Rules (FSR) allows developers to write JSON documentations which define exactly what data is accessible.

FSR will be used to define explicit conditions and criteria which must always be met before a request is made to our database. The Request's path will be processed and inspected until confirmed that it is not a user with malicious intent. Any requests that are matched will only have access to the intended documents and not the entire collection of data we are protecting. For example, a company rep wishing to access a student's CV or academic record will only be able to download that student documents and not the entire database of data.

Additionally, we will work closely with the Authentication Team to ensure that any user attempting access to the database will be identified through their UserID, Credentials and/or authentication tokens. Thus, a student/company that is not signed into an account with the designated privileges will be barred from accessing any documents in our storage system.

## Contracts



The screenshot displays the Google Cloud Storage console interface. At the top, the bucket name 'COS301 Storage Mini-Project' is visible. Below the bucket name, the 'Storage' section is active, showing a table of folders. The table has columns for Name, Size, Type, and Last modified. The folders listed are BlogMedia/, DatabaseDumps/, Files/, ProfilePhotos/, Thumbnails/, and Videos/. A blue 'Upload file' button is located in the top right corner of the table area.

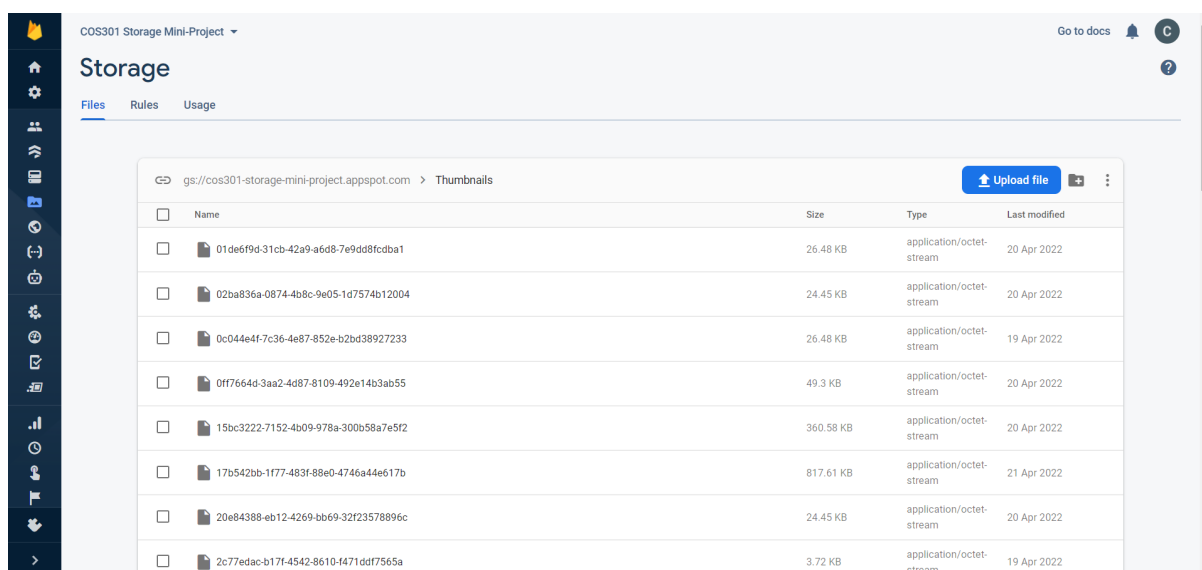
Name	Size	Type	Last modified
BlogMedia/	-	Folder	-
DatabaseDumps/	-	Folder	-
Files/	-	Folder	-
ProfilePhotos/	-	Folder	-
Thumbnails/	-	Folder	-
Videos/	-	Folder	-

## Storage API & Company Representative Profile

An integration exists between our Storage API and the Company Representative Profile team which uses our API to store the user's profile photos. This is done through calling our API, which already has the functionality implemented to communicate with our database. The profile photos are then stored on our file base.

## Storage Firebase Repo & Hosting/Shorts/Blog

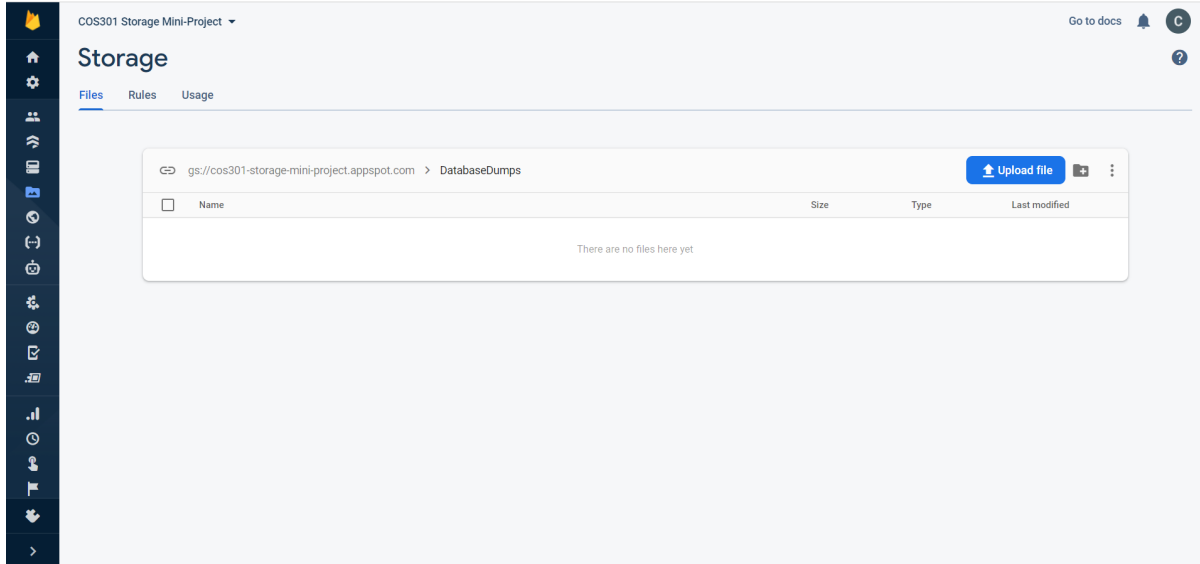
The Firebase Repository of our Storage feature was made available for the Hosting, Shorts and Blog features to store any documents/images/videos etc. Although the storage falls under our repository, we do not monitor or are involved with the design of schemas or databases these features have implemented in this repository.



## Storage Functions & Hosting Feature

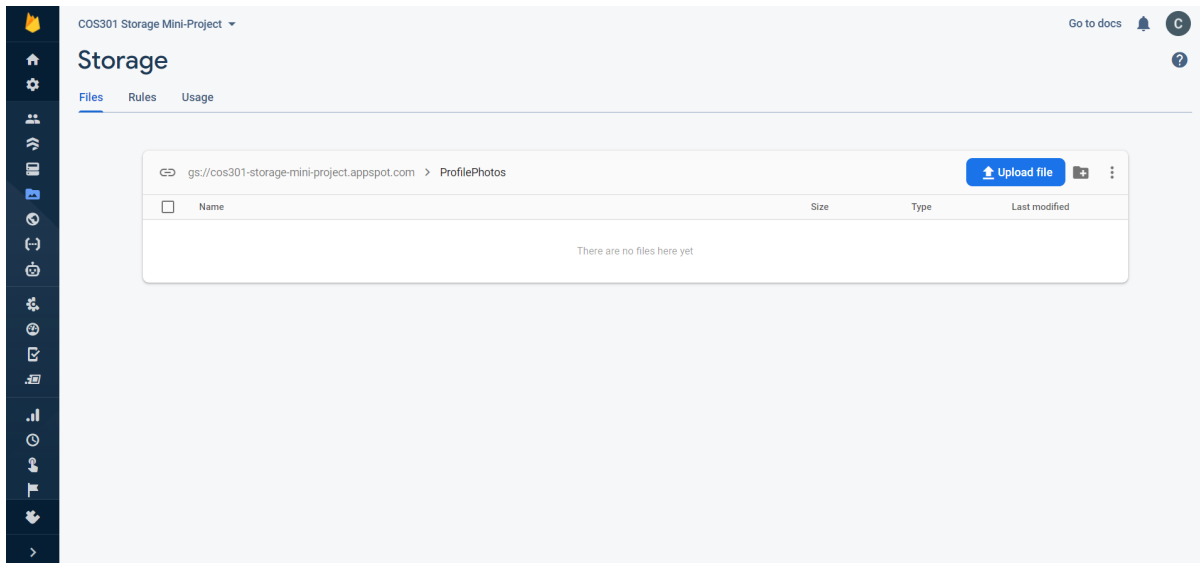
Our data engineer implemented a directory upload function specifically for the Hosting feature. This function allows the Hosting team to upload any Database Dumps created from their directory. The database dumps will be stored in our Storage firebase and will be accessible and retrievable by the Hosting Team.





## Storage Repository & User Profile

The User Profile feature makes use of our storage Repository which needed to be altered for this integration. This was implemented to allow for profile photos to be stored in our relation but in a different file category. The profile photos stored will be accessible and retrievable by the User Profile Team.



## Acceptance Criteria

The storage feature provides a backbone for the entire Mini-Project as a core functional requirement is data upload/retrieval by the respective users. With this fact in mind, we have set mandatory acceptance criteria before the release of this feature:

- File upload must have a near perfect success rate, when users use our feature there must be minimal errors when interacting with the database. Any errors while uploading must be caught and the user must be notified. [Signed by Christian Devraj - Business Analyst of Team FORTRAN]
- File download must be quite reliable as well. When a user attempts to download the file it must be:
  - The correct file selected by the user. [Signed by Christian Devraj - Business Analyst of Team FORTRAN]
  - The same file format that was uploaded. [Signed by Christian Devraj - Business Analyst of Team FORTRAN]
  - Not corruptible in any way. [Signed by Christian Devraj - Business Analyst of Team FORTRAN]
- The UI must be aesthetically pleasing and completely functional, a UI with even a few faults will make the user-experience inadequate. This in turn will damage our entire system's reputation among other repercussions. [Signed by Christian Devraj - Business Analyst of Team FORTRAN]
- Features should be able to integrate the other features and the mini-project as a whole. [Signed by Christian Devraj - Business Analyst of Team FORTRAN]