# Functional Requirements Document

# General

## Project Description

The IPFS Transfer API provides an easy way to upload multiple files along with accompanying file data to be hashed using the ipfs-api and to be stored in a database.

### Background

The need for this API arose to save server space in applications where the client may upload multiple files for long term storage.

### Purpose

The purpose of this API is to content-based hashes for files using the ipfs-api so that they can be accessed in the future using only the hash, and thus saving considerable server space.

### Assumptions and Constraints

Assumptions:

* Availability of all the required software on the server where the API is deployed
* Existence of at least one copy of the files being hashed on the network
* Security measures are taken by the person or organization using this API to protect their files

Constraints:

* The time required to hash files is constrained by the ipfs-api and is outside of the developers’ control
* The deletion of a file permanently from the IPFS would require every copy of the file on the network to be destroyed

### Interfaces to External Systems

the applications with which the subject application must interface, State the following for each such application:

* Client application – This is only a server side application and as such it will require a client side application for most use cases
* For details on how to interface with this application, refer to the Swagger documentation at ‘/api-docs/v1.0/’

## Points of Contact

* Project Manager: Kuldeep Narvekar E: kuldeep@cateina.com
* Developers: Mohammed Burk E: [mohammedburk@gmail.com](mailto:mohammedburk@gmail.com)

Aditya Vaish E: [adityavaish5@gmail.com](mailto:adityavaish5@gmail.com)

Mahir Patrawala E: mahirpatrawala99@gmail.com

* Agency: Cateina Technologies

## Document References

Name the documents that were sources of this version of the FRD. Include meeting summaries, white paper analyses, and other System Development Life Cycle deliverables, as well as any other documents that contributed to the FRD. Include the Configuration Management identifier and date published for each document listed.

# FUNCTIONAL REQUIREMENTS

The functional requirements describe the core functionality of the application. This section includes the data and functional process requirements.

## Data Requirements

1. /upload: Requires files to upload, database information and form data (optional).
2. /returnFiles: Requires database information.
3. /updateFiles: Requires object ID of the file to be updated, new file, database information and form data (optional).
4. /deleteFile: Requires object ID of the file to be deleted and database information.

## Functional Process Requirements

The API has the following functionalities built into it:

1. Upload files: Allows the user to upload multiple files along with accompanying form data. The file is hashed using the ipfs-api, bound with the respective form data and then stored in the database.
2. Return files: Returns all the file and form data stored in the database.
3. Update file: Updates the contents stored at a particular object ID.
4. Delete file: Deletes the contents stored at a particular object ID.

# OPERATIONAL REQUIREMENTS

## Security

There is no built-in authentication in this API and access permissions are dependent on the person or organization using this API.

The consequences of not providing security measures may include loss of data or disclosure of privileged information.

## Reliability

The failure of the ipfs-api may cause the user unable to access the file using the hash.