UNISTAD Document Manager

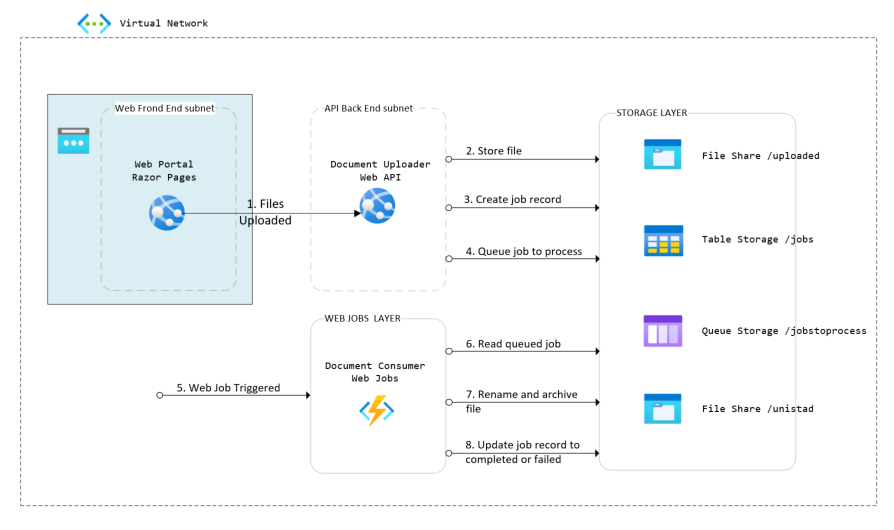
# Overview

This Web Application will be responsible to organize automatically the UNISTAD documents in Azure Storage File folders and apply the naming and convention defined for the files.

The application contains the following components:

* Document Uploader: Web API which receives the document uploaded to be processed.
* Document Consumer: Azure Function which stores the document uploaded in the correct Storage File Folder.
* Web application: Web page with the interface to upload the files.

# Architecture



# Document Uploader – Web API Specification

## API CREATE Operation

Creates the file uploaded in the File Storage. There is a default folder where the files uploaded are stored (see topic Storage Entities -> File Storage for details about the folders). Create a record in the table storage to track the status of the job. Enqueues a message in the Storage Queue indicating the file was uploaded, so the job needs to be processed by the Web Jobs.

|  |
| --- |
| **HTTP POST** |
| https://[servicename].azurewebsites.net/api/v1/uploadedimage?fileName=SC-I60-CAB-ORD-DBF-IT-00102.pdf |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **URI Parameters** | | | | | |
| Name | Required | Data Type | Length | Constrains | Description |
| fileName | True | String | 200 | Valid file name | File name |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Request Body** | | | | | |
| Name | Required | Data Type | Length | Constrains | Description |
| fileData | True | File | n/a | Not null | The file upload data  .NET Type: **IFormFile** |

Additional Requirements:

* + 1. The content type shall be automatically set using the content type provided by IFormFile fileData parameter.
    2. The id assigned to the Job shall be generated automatically by your implementation. You shall use a Guid converted to a string for the Job’s id. This id will be added in the Table storage RowKey column and also in the Queued message.
    3. Each file uploaded will be considered a Job.
    4. See Topic Storage Entities to get all details about the File Storage, Table Storage, and Queue Storage.

|  |
| --- |
| **Response 201 (Created)** |
| Returned if the file was stored in the uploaded folder successfully. Table job row was created, and a message was added to the Queue.  **Headers**  **Location: contains the storage address to the file uploaded, example** [https://**[service**](https://[service) **name]**.azurewebsites.net/api/v1/[uploaded folder name]/[file name uploaded]  See Storage topic for further details about the folders name used for the uploaded files.  **Body**  Empty (no content) |

|  |
| --- |
| **Response 400 (Bad Request)** |
| Returned if the request has no file data or already exist a file in the uploaded folder with the same name.  **Headers**  None, beyond standard  **Body**  Return an error response using the ErrorResponse JSON format, see topic about the Error Response topic for further details. |

# Document Consumer – Web Jobs

This WebJob is triggered when a new message is queued. The WebJob will be responsible to scan the document, identify all elements which produces the correct final file name, if all elements are found the file is moved with the correct name to the correct folder.

Once the operation is completed, all details are updated in the record in the Storage Table associated to the file (or message).

# Storage Entities

## Table Storage Specification

Table storage name will be **unistadjobs**, If the queue **unistadjobs** does not exist, then it shall be created.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Column | Data Type | Length | Description | Value |
| PartitionKey | String | 20 | Project name | UNISTAD |
| RowKey | String | 36 | The id of the job. | The id will be a Guid. |
| Status | Number | n/a | A number indicating the status of the job  1 = File uploaded  2 = Job is running  3 = Job completed with success  4 = Job failed | 1 |
| StatusDescription | String | 512 | If no error occurred this value contains the text associated with the status defined above.  Ex: File uploaded.  If an error occurred a human readable description of the problem running converting the image. Always prefix this with "Job failed:" followed by a more descriptive message. This can be the message from an exception that occurred but NOT THE STACK TRACE.  Ex: Job failed: The uploaded image could not be converted. | File uploaded |
| FileSource | String | 255 | Original name of the file. | SC-I60-CAB-ORD-DBF-IT-00102.pdf |
| FileResult | String | 255 | The path where the file was stored and the new file name.  In case of any failure, the file is moved to the failed folder. | /HB/Package 1/Base/MS – 2/HB-ACS-HLFD- SC-I60-CAB-ORD-DBF-IT-00102.pdf  Or  /\_Failed/ SC-I60-CAB-ORD-DBF-IT-00102.pdf |

## Queue Storage Specification

Azure Storage Queue shall be called: **unistadprocessqueue,** If the queue **unistadprocessqueue** does not exist, then it shall be created.

The message will have the following JSON format:

|  |
| --- |
| {  “PartitionKEy” : “UNISTAD”  “RowId” : “[Guid]”  “FileName” : “[Original file name]”  } |

RowId is the link with the Jobs table.

FileName is the name of the file stored in the uploaded folder.

When the process is completed, the message is removed from the queue.

## File Storage Specification

Azure Storage File will have the following folders:

* /jobs/uploaded : Folder where the files uploaded are stored temporally.
* /jobs/failed : Folder where the files, where the consuming process failed are stored.
* /unistad : Root folder where the files consumed will be stored.

Additional Requirements:

* + 1. The unistad folder will be organized using the following sub folders: /unistad/[Stadium]/[Package + number]/[\*Release name]/[Milestone number].
       - The “\*” means optional, as for Package 2 there are no releases at the moment, so this is omitted.
       - Packages names are: Package 1 or Package 2.
       - Release names are Base, Release 1, Release 2.
       - Milestones numbers are MS – 1, MS – 2, MS – 3, …, until MS – 6.
    2. When file is saved in the **target folder**, in case the file already exists the job will be considered failed, the file will be stored in the filed folder.
    3. When file is saved in the **failed folder**, the last 4 characters of the job are added as suffix of the file, as the same file name can be uploaded several times.

# Error Response JSON

This is the standard error response that shall be returned.

|  |
| --- |
| {  "errorNumber":<error number>,  "parameterName":"<name of parameter that caused the error>",  "parameterValue":"<value of parameter that caused the error>",  "errorDescription":"<Description of the error intended developer consumption>"  } |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | Required | Data Type | Length | Description |
| errorNumber | True | number | n/a | Numeric error that represents the issue  .NET Type: int |
| parameterName | False | String | 1024 | The name of the parameter that has the issue.  If the error is not tied to a specific parameter, then this value can be null  .NET Type: string |
| parameterValue | False | string | 2048 | The value of the parameter caused the error  If the error is not tied to a specific parameter, then this value can be null  .NET Type: string |
| errorDescription | True | String | 1024 | A description of the error, not localized, intended for developer consumption  .NET Type: string |

# API Error Response Codes

Below are the common codes and error descriptions to be used by the APIs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **errorNumber** | parameterName | parameterValue | errorDescription | Notes |
| 1 | Required | Required | The document was already uploaded. | This error is raised when file is uploaded. |
| 2 | Required | Required | The parameter is required. |  |
| 3 | Required | Required | The entity could not be found. |  |
| 4 | Required | Null | The parameter cannot be null. |  |

Below are the common codes and error descriptions to be used when consuming the document uploaded.

|  |  |  |
| --- | --- | --- |
| **errorNumber** | errorDescription | Notes |
| 21 | The stadium name wasn’t found. | Raised when searching the **name of the stadium** in the pdf. |
| 22 | The service name wasn’t found. | Raised when searching the **name of the service** in the pdf. |
| 23 | The document type wasn’t found. | Raised when searching the **document type** in the pdf. Examples: Business Requirement, High Level Functional Document, etc. |
| 24 | The document reference number wasn’t found. | Raised when searching the **document reference number** in the pdf. Example : SC-I60-CAB-ORD-DBF-IT-00102 |
| 25 | The version wasn’t found. |  |
|  |  |  |
| 25 | File already exists. | When the file result already exist in the folder. |

# Log Error Codes

This is the list of errors listed in the log. The codes follow this pattern :

100-149 : Errors at the UnistadDocumentLibrary Project.

200-249 : Errors at the DocumentConsumer Project.

250-299 : Errors at the StorageLibrary Project.

| **Error Number** | **Location** | **Raised When** | **Cause** |
| --- | --- | --- | --- |
| 114 | Unistad Document Library :  UnistadDocument Class | Not found in the pdf the following information: Stadium, Service, or Document Type. The application is not able to work out the document name. | Document is not an UNISTAD document managed by the application. Ex:PM & DM Plan.  Missing a configuration related to the mapping between value and code in the host.json. |
| 115 | Unistad Document Library :  UnistadDocument Class | Not able to find the mapping between the code (stadium, service, document type) and target folder or sub-folder to store the file.  When 114 is raised 115 is raised also as the code wasn’t found | When error 114 is raised, the error 115 is raised also.  Missing a configuration related to the mapping between code and folder location in the host.json file. |
| 200 | DocumentConsumer : DocumentConsumer Class | Unknown error raised when UnistadConsumer function was processing the file provided in the triggered queued message. | Check it out the log. More details of this errors are recorded. |
| 201 | DocumentConsumer : DocumentConsumer Class | When archiving the file at the target folder, a file with the same name already exists in the target folder. In this case the process fails, the file being processed is stored in the Failed folder, using the original name + 5 last GUID letters. | Same file was already uploaded before.  Application wasn’t able to define correctly the name of the file. Possible cause can be the version or service name. |
| 251 | StorageLibrary : Repositories. FileSystem | The file to be deleted or moved can’t be found.  I don’t believe this error will happen, but when removing the file from the Source Uploaded Folder after being process, the file wasn’t found. | The file was removed by another process.  The folder or file was renamed while file was being processed.  Please analyze carefully as this error wouldn’t happen. |
| 253 | StorageLibrary : Repositories. FileSystem | Unknown error moving the file. | Check error log for further details. |

# Class Unistad Document

Class responsible to implement all operations needed to rename the original uploaded file to the correct naming and convention, it also has the responsibilities to implement the methods required to store the file in the correct folder. This class will make use of the package which implements all operation into pdf file.

Constructor

* UnistadDocument (FileSourceName, FileShareClient)

Methods:

* Private GetStadiumCode
* Private GetServiceCode
* Private GetDocumentTypeCode
* Private GetDocumentReference
* Private GetDocumentName
* Private GetDocumentFolder
* Public StoreDocument