**Technical Solution Approach**

Contents

[1 Introduction 2](#_Toc127885736)

[1.1 About this document 2](#_Toc127885737)

[1.1.1 Purpose & Scope of the document 2](#_Toc127885738)

[2 Component Design 2](#_Toc127885739)

[2.1 Component Design Diagram 2](#_Toc127885740)

[2.1.1 Overall Workflow 2](#_Toc127885741)

[2.1.2 Low level Design 2](#_Toc127885742)

[3 Technology & Frameworks used 3](#_Toc127885743)

[4 Solution Approach 3](#_Toc127885744)

5 Test Case Scenario…………………………………………………………………………………………………………………….3

# Introduction

## About this document

### Purpose & Scope of the document

The purpose of the component specification is to systematically record the understanding of component design and the expected usage. This document also includes the design elements and the workflow wherever possible.

# Component Design

## 2.1 Component Design Diagram

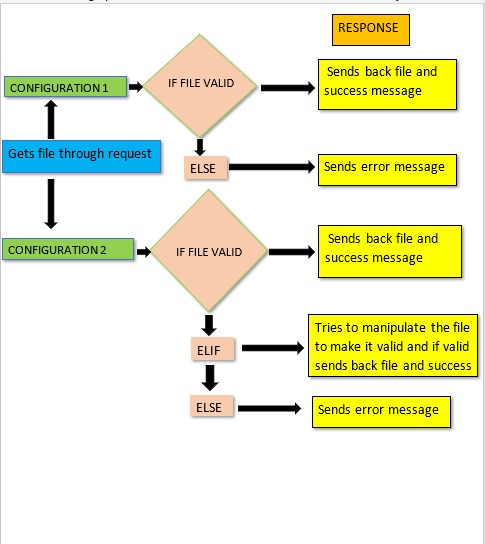
### Overall Workflow



E.g. Sample diagram below is for the File Validation.



### Low level Design



# 

# Technology & Frameworks to be used

1. Fast API

2. Python

3. Pandas

4. Polars

5. Duckdb

# Solution Approach

WHY API?

We started by working on creating a library but decided to work with an API because it can implemented in any techstack. We didn’t work on database connectivity because we figured our API could be implemented directly into the workflow.

The API can be customised using a configuration file and is made to validate files. It has three endpoints, each with a unique setup, and logs each entry in a DuckDB file. Both the first endpoint and the second endpoint validate the file using configuration 1 and 2, respectively. For each endpoint, the configuration settings are defined in order to tailor the validation procedure. The API offers a versatile and effective way to validate files with various setups and log each entry for additional investigation.

The file validation process involves four steps:

1. User uploads file in frontend: The user selects a file from their device and uploads it to the frontend.

2. Frontend hits API with file: The frontend sends a request to the API along with the uploaded file.

3. API validates the file: The API receives the file and validates it according to its rules and criteria. If the file is valid, it returns True with the file. Otherwise, it returns False.

4. Returns result: The API sends the validation result back to the frontend, which can display a message indicating whether the file is valid or not.

Overall, this process provides a simple and efficient way to validate files uploaded by users, ensuring that only valid files are accepted for further processing.

Note: Regardless of the files status every entry is logged

# Test Case Scenario:

(NOTE: error messages are configurable)

Test case 1:

Validating the file that is within the size limit

Input: uploading the file that’s within the size limit (10mb)

Output: Returns status = 1 and file

Test case 2:

Validating the file that is above the size limit

Input: uploading file with size limit above the given constraint (above 10mb)

Output: returns status = 0 and error message

Test case 3:

Validating a corrupted file

Input: uploading a file that is corrupted

Output: returns status = 0 and error message

Test case 4:

Validating an uncorrupted file

Input: uploading a file that is uncorrupted

Output: Returns status = 1 and file

Test case 5:

Validating a file with correct format

Input: uploading a file that is written in a right format (satisfying all the conditions)

Output: Returns status = 1 and file

Test case 6:

Validating a file with incorrect format

Input: uploading a file which is written in wrong format (not satisfying all the conditions)

Output: returns status = 0 and error message

Test case 7:

Validating a non-empty file

Input: uploading a non-empty file

Output: Returns status = 1 and file

Test case 8:

Validating an empty file

Input: uploading an empty file

Output: returns status = 0 and error message

Test case 9:

Validating a file with valid column length

Input: uploading a file with valid column length (20columns)

Output: Returns status = 1 and file

Test case 10:

Validating a file with invalid column length

Input: uploading a file with invalid column length (more than 20columns)

Output: returns status = 0 and error message

Test case 11:

Validating files with date column that is in correct format

Input: uploading a file with date column that is written in correct date format

Output: Returns status = 1 and file

Test case 12:

Validating a file with date column that is in incorrect format

Input: uploading a file with date column that is written in incorrect date format

Configuration\_1:

error message

Configuration\_2:

Changes the incorrect date format to correct date format

Output: Returns status = 1 and file

Test case 13:

Validating a file with unique unique columns

Input: uploading a file with unique columns being unique

Output: Returns status = 1 and file

Test case 14:

Validating a file with no unique unique columns

Input: uploading a file with unique columns being not unique

Output: Returns status = 0 and error message

Test case 15:

Validating a file with valid column data types

Input: uploading a file with columns written in valid data types

Output: Returns status = 1 and file

Test case 16:

Validating a file with invalid column data types

Input: uploading a file with columns written in invalid data types

Configuration\_1:

Error message

Configuration\_2:

Tries to change the columns data types to desired ones

If Valid:

Output: Returns status = 1 and file

Else:

Output: Returns status = 0 and error message

Test case 17:

Validating a file with mandatory column

Input: uploading a file with mandatory column

Output: Returns status = 1 and file

Test case 18:

Validating a file without mandatory column

Input: uploading a file without mandatory column

Output: Returns status = 0 and error message

Test Case 19: Validating file upload for exact duplicate records

Input: Upload a file with exact duplicate records

Output: API eliminates exact duplicate records and return status=1 and file

XML SPECIFIC TEST CASES:

Test case 1:

Validating a file without multivalued columns

Input: uploading a file without multivalued columns

Output: Returns status = 1 and file

Test case 2:

Validating a file with multivalued columns

Input: uploading a file with multivalued columns

Configuration\_1:  
Error message

Configuration\_2:

removes multivalued column tags and validates the file

If Valid:

Output: Returns status = 1 and file

Else:

Output: Returns status = 0 and error message

Test case 3:

Validating a file without multiple tables

Input: uploading a file without multiple tables

Output: Returns status = 1 and file

Test case 4:

Validating a file with multiple tables

Input: uploading a file with multiple tables

Evaluates each table individually

If all tables are valid:

Output: Returns status = 1 and file

Else:

Output: Returns status = 0 and error message