SOLUTION APPROACH DOCUMENT

FILE UPLOAD IN ANGULAR 13 AND SPRING BOOT JAVA

Report Submitted

by

SAI SRINIVAS VARA PRASAD KORLAM	(VTU17342) - Development (Backend)
SRIPATHI MNIDEEP REDDY	(VTU17713) - Development (frontend)
RISHIKA KARUTURI	(VTU18063) - Design and Planning
VASANTHI DEVI P	(VTU18041) - QA Engineer
GAURAV VITRAG	(VTU17236) - Test Engineer

Under the guidance of

Dr. R. ARUNA (Associate Professor)

Ms. M. MEENALAKSHMI (Assistant Professor) Mr.

N. SIVA RAMA LINGHAM (Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN Dr. SAGUNTHALA R&D INSTITUTE OF SCIENCE AND TECHNOLOGY

ABSTRACT

With the modern digitization of the technology, a lot of upload systems have come to light. Even with such introduction of systems, there exist an acute need of analytical upload systems with user-specific control conditions. Recently the concept of file uploading has become very common in our environment, but it does not have further functions apart from accepting file and storing it in the local drive or the database. This activity discusses and implements a file upload and validation system for achieving this target. This file upload component is a library that makes file uploading much more flexible to all applications with slight code shifts. This component is built for re-useability and generic functionality. The angular framework is used to design the single dynamic page application accepts file which has sizes lesser than the limit and only of format CSV, XLS, XLSM and XLSX. It also gathers the module name from the user where it needs to be stored ina database. The file will be transferred to the back-end spring boot service through the HTTP module using the secure POST method. Springboot is a Standalone application with an embedded server. It read those file and check for constraints acceptance. After satisfying those constraints then the constraints acknowledgement is sent to the user. Then it processes the file for compatibility and stores the file on a local and MongoDB database as a record within the module.

Keywords: Angular, Spring Boot, HTTP module, POST method, MongoDB

LIST OF FIGURES

ABBREVIATIONS

CSV	Comma Separated Value
XLSM	Excel Macro-Enabled Workbook
XML	Extensible Markup Language
JSON	JavaScript Object Notation
SQL	Structured Query Language
NPM	Node Package Manager
HTTP	Hyper Text Transfer Protocol

TEAM ROLES

Roles	Member Name
Design and Planning	Rishika
QA Engineer	Vasanthi devi
Development (Backend)	Sai Srinivas Vara Prasad Korlam
Development (frontend)	Manideep Reddy Sripathi
Test Engineer	Gaurav Vitrag

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

1.1 ABOUT THIS DOCUMENT

1.1.1 PURPOSE & SCOPE

- The main purpose of this component is to upload files based on the file size and file format like CSV, XLSM, XLSX and XML.
- It also stores those files in the database after removing unwanted columns, duplicate records, and excess columns, and verifying data types based on the config file.
- The scope of building this component is reusability and generic which can be later integrated into any application of the same type or cross-application with minor changes.

1.1.2 OVERVIEW

This file upload component is a generic and reusable library which makes it more flexible to all applications. The angular framework is used to design the single dynamic page application with file upload page used as a front end which serves as a UI component. It also gathers the module name and file. That file will be transferred to the Sprint Boot is a Standaloneapplication with an embedded server. It read those files and check for constraints. When theconstraints are satisfied then the acknowledgement is sent to the user and the file will be stored in the database as a record within the module. This document serves as guide for thedeveloper who looks to use the component without any difficulties.

CHAPTER 2

COMPONENT DESIGN

2.1 COMPONENT DESIGN DIAGRAM

This section describes about the component design and the specific way of designing.

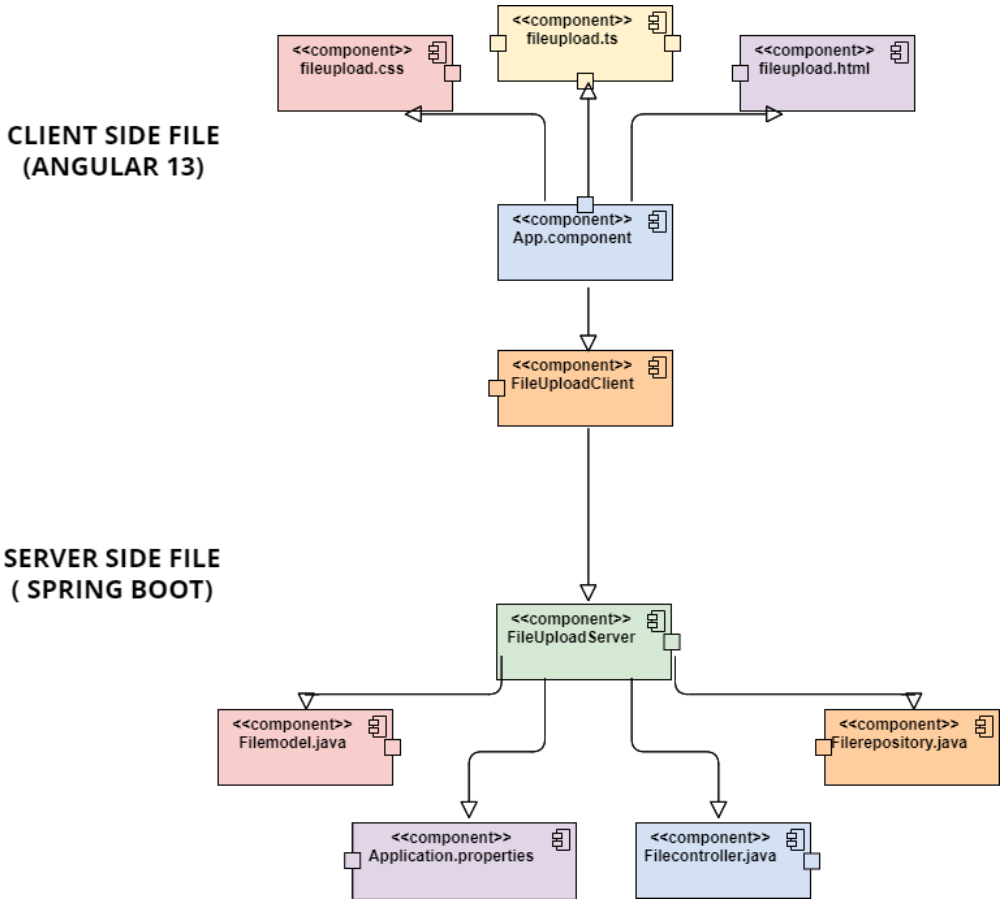


Fig 2.1: *Component Diagram of File Upload*

On client-side, the application uses the combination of angular component with html and CSS as user interface.

App.component : This is the main component from where it starts to implement. It calls the Fileupload component.

FileUpload.html : The html page by which the static page of the UI interface created.

FileUpload.css : The CSS stylesheet helps design the element which are available in the FileUpload component.

FileUpload.ts : The main part the component from where the connection, validation and transferring of file to server is done.

On server-side, the application uses combination of spring boot to extract file and store in database.

Application.properties : This is location where all property of the rest control application is stored and used for needs like running URL, database username and password,

Filecontroller.java :It create the controller which can gather the file from the URL by mapping and return the acknowledgement.

Filemodel.java : Model file contains the data of the specific application, the data is json object.

Filerepository.java : Repository file provide the mechanism of storage, retrieval, search and database operation.

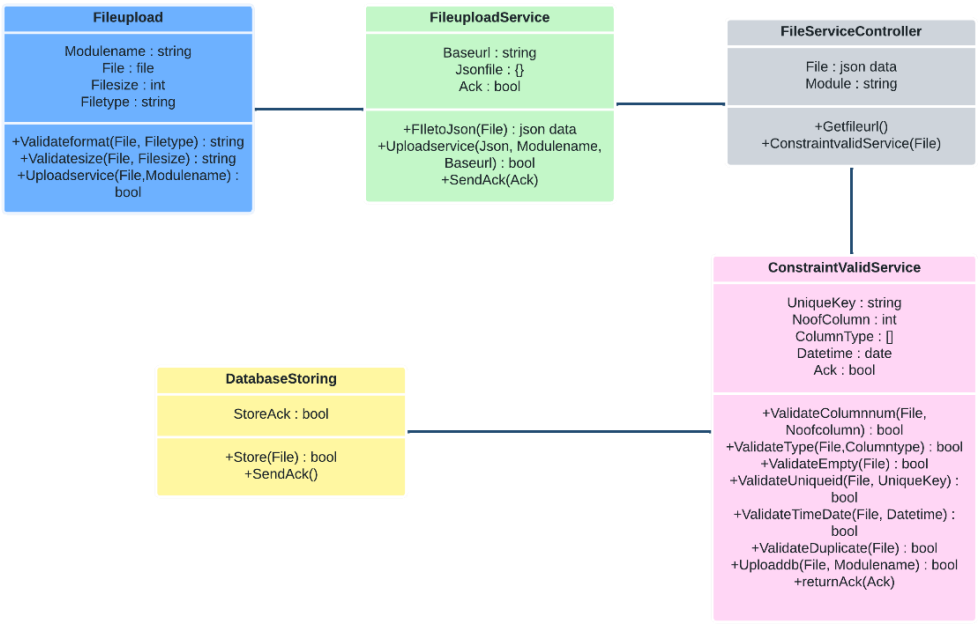


Fig 2.2: *Class Diagram of File Upload*

The front-end processes in this case, which include the file size and file type, are handled bythe file upload. This verifies the file size and format and gives the user an acknowledgment.Finally, in the Validate and store () function, we are required to use the File Upload Controller object to submit a POST request. This object belongs to the main class library object that has beenbuilt, which is File Upload Service. Every class method will be verified in this. Next, we updatedthe File Upload Controller Class with the success/failure message. Using the same data() functionof the db_context class, we store the message if it is successful. the message follows

2.1.1 OVERALL WORKFLOW

This component workflow diagram describes the complete working process of the interdependent component. Initially, the user uploads the file which gets validated. Then it passes to the Spring Boot service. Then service will receive the file and check for constraints and store it in the database.

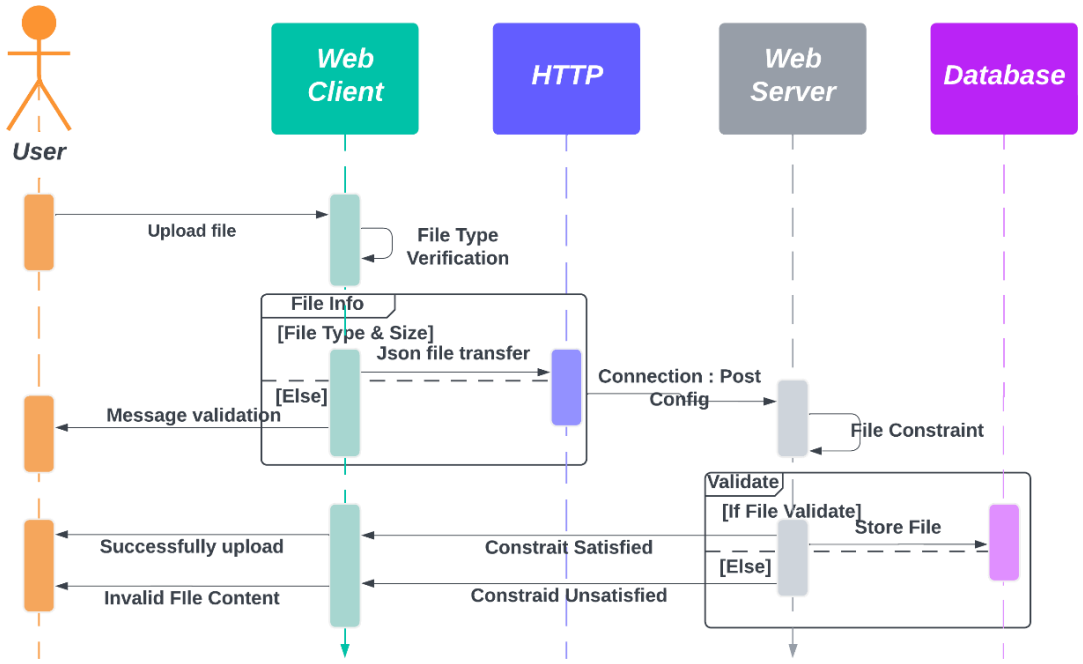


Fig 2.3: *Sequence Diagram of File Upload*

● The sequence diagram shows how to upload a file from a web client to a web server and store the information in a database. The web client, web server, and database are all elements of this process.

● The online client will ask the user to upload a file at the start of the procedure, which can be in Excel, CSV, or XML format. The file is sent from the client to the web server when the user picks it.

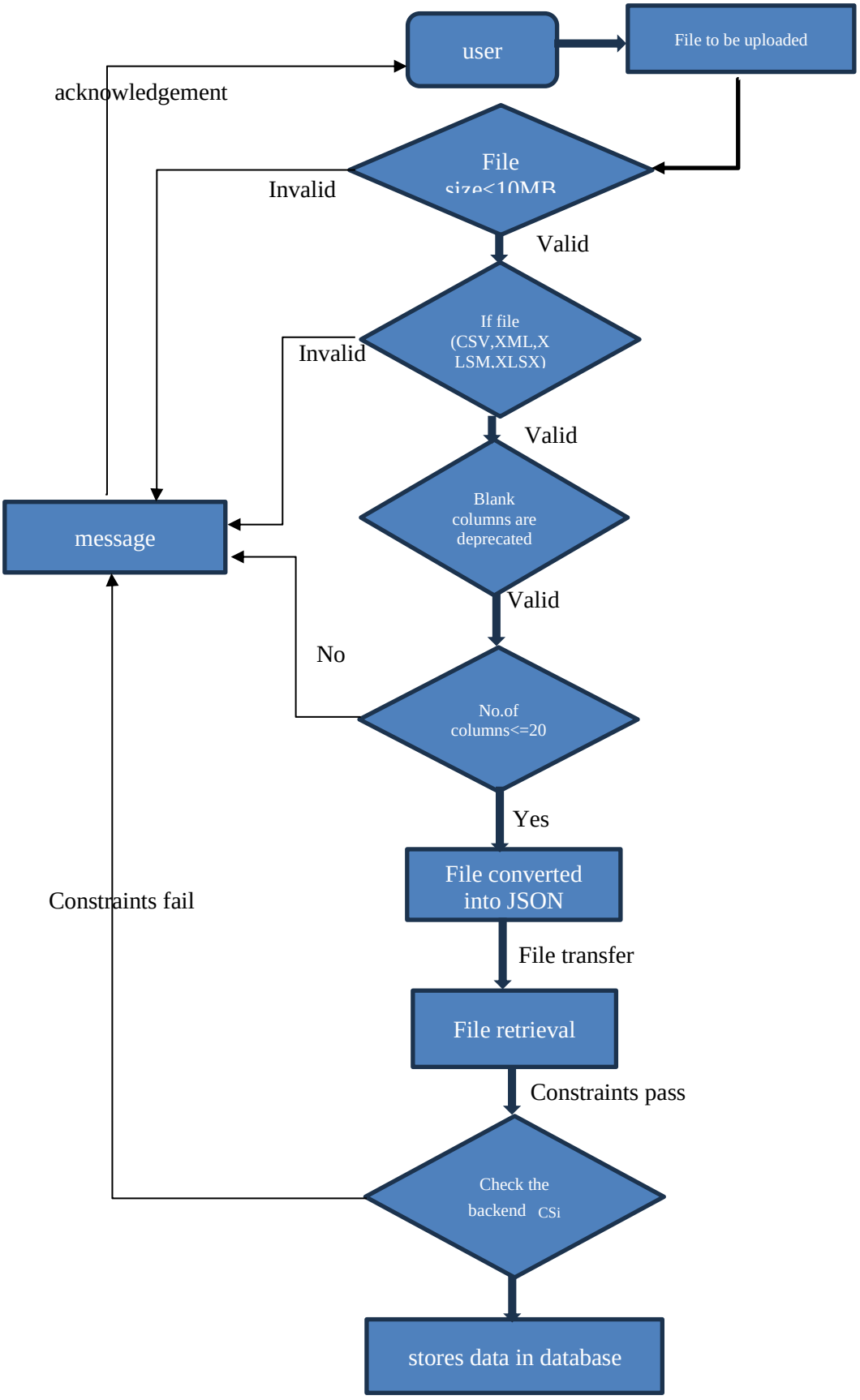
● For instance, the server could confirm that the file size is within a predeterminedrange, the file format is accurate, and that the necessary fields are present.

● If the validation checks fail, the server informs the web client that the file upload was unsuccessful by issuing a failure message. In the event that the validation checks are successful, the server stores the JSON data in the database.

● The server then notifies the web client that the file upload was successful by returning a success message. This message may be used to alert the user that the data has been successfully uploaded and saved in the database and to refresh the user interface.

● In summary, the sequence diagram depicts the procedures involved in uploading a file, verifying the data, and saving it in a database. This procedure is required for many web applications that need users to submit data, such as social networking platforms, e-commerce websites, and data analytics tools.

2.1.2 LOW LEVEL DESIGN



CHAPTER 3

TECHNOLOGY & FRAMEWORK

3.1 FRONT-END

3.1.1 ANGULAR

Version 13

- Angular is an application-design framework and development platform for creating efficient and sophisticated single-page apps.
- It is a component-based framework for building scalable web applications using typescript.
- A suite of developer tools to help you develop, build, test, and update your code.

Purpose

- To create a dynamic page for authentication and file upload. It validates the file format and size and returns the acknowledgement to the user.
- It passes the file to the service by converting it into a JSON file with the module name.

Prerequisite

Node js

Version 19

- Node js is the open-source JavaScript environment used to load all the dependencies for angular applications and it acts as a web server.

Npm

Version 8

- Node Package Manager is installed along with node js. Npm is a default package manager for JavaScript.
- Npm manages the JavaScript runtime.

3.2 BACK-END

3.2.1 SPRING BOOT

Version 3

- Spring Boot helps to create apps that are not platform specific and that can run locally on a device without an internet connection or other installed services to be functional.
- It is a standalone application with an embedded server.

Purpose

- Spring Boot service will gather the JSON file along with the module name from Angular through the POST method.
- It will check for constraints and processes them before storing them in the database.

Prerequisite

Maven

Version 3.9

- The Spring Boot Maven plugin provides support for Spring Boot in Apache Maven.
- It packages all executable jar and war archives and helps to run the Spring Boot application.
- It generates build information and boots the Spring Boot application before running tests.

Tomcat

Version 10

- Tomcat is a web container that manages multiple applications and avoids each separate application setup.
- It makes the application run on the web server.

3.3 TOOLS USED

3.3.1 VISUAL STUDIO CODE

- Visual studio code is a code editor with all language support functionality. It supports development processes like debugging, version control, and testing.
- No separate framework or packages are needed, it can automatically plugin all libraries and frameworks.

3.3.2 ECLIPSE

- Eclipse is an effective integrated development environment (IDE) that programmers use to easily write, debug, and manage code.
- It has capabilities including code editing, debugging, and project organisation, all of which are supplemented by a sizable plugin ecosystem. It encourages cooperation and provides strong communal backing.

3.3.3 POSTMAN

- Postman is a flexible API testing and development tool that makes it easier to send requests, receive responses, and debug APIs. This aids developers in efficiently streamlining their API workflows.
- Postman gives developers the tools they need to efficiently construct, test, and document APIs thanks to its user-friendly interface and wealth of capabilities

3.3.4 SONARQUBE

- SonarQube is a potent code quality management tool that examines source code for errors, weaknesses, and code smells and gives developers useful information to enhance the quality of their product.
- SonarQube assists teams in ensuring code integrity and upholding high standards in their projects with its comprehensive reporting and continuous integration features.

CHAPTER 4

SOLUTION APPROACH

• APPROACH DESCRIPTION

This section provides a detailed view of how the components are partitioned and assigned to the functionalities.

1. The user selects a file of types of Excel, CSV, or XML from their local file system and clicks on the upload button. Then it checks for the constraints.
2. if validation fails, the corresponding validation message is sent back to the front-end.
3. In case all validations pass, the data is uploaded into the database, and a Success message is returned to the front-end. The creation of RESTful services for handling HTTPrequests sent from the AngularJS front-end can be achieved using Spring Boot. Service methods can be implemented using Spring libraries, such as Spring Data JPA and Spring MVC, which are responsible for handling file upload and JSON data conversion.
4. Validations - The file upload functionality has the following constraints:
    - Files must not exceed 10 MB.
    - File should support a maximum of 20 columns data.
    - The allowed file format should be XLSX, XLSM, CSV, and XML.
    - The system should ignore blank columns that are between the columns having the data.
    - The system should be able to check whether the columns are mandatory or not based on the mandatory configuration done for each column in the config file
    - Unique Key column, which is defined in the config file, should not be duplicated.
    - The system should be able to eliminate exact duplicate records before bulk insert or update. It undergoes unit and functional testing. Then detect the issue caused and fixed.
    - The system should validate the column length, which is defined in the config file.

- The system should validate the data type of each column, which is defined in the config file.
- The system should be able to validate whether the date/date-time column adheres to a specific date/date-time format (i.e., MM/DD/YYYY or DD/MM/YYYY, etc.), which is configured in the config file.

5. The application should handle exceptions caused by timeouts or other failures by logging them in the application logs. If the exception is due to incorrect data or format, the application should send back user-friendly messages to the UI. Spring Framework libraries such as Exception and Spring Boot Logging can be used for exception handling, while Jackson can be used for serializing and deserializing JSON data.
6. Customized order of validation to ensure optimum flow of acceptance:

C_1: The allowed file format should be XLSX, XLSM, CSV, and XML.

C_2: The system should ignore blank columns that are between the columns having the data.

C_3: File should support a maximum of 20 columns.

C_4: The system should be able to check whether the columns are mandatory or not based on the mandatory configuration done for each column in the config file.

C_5: Files must not exceed 10 MB.

C_6: Unique Key column, which is defined in the config file, should not be duplicated.

C_7: The system should be able to eliminate exact duplicate records before bulk insert or update. It undergoes unit and functional testing. Then detect the issue caused and fixed.

C_8: The system should validate the column length, which is defined in the config file.

C_9: The system should validate the data type of each column, which is defined in the config file.

C_{10}: The system should be able to validate whether the date/date-time column adheres to a specific date/date-time format (i.e., MM/DD/YYYY or DD/MM/YYYY, etc.),which is configured in the config file.

TEST CASE SCENARIO

- Test Case 1: Validating file upload for correct file type Input: Upload a file with correct file type (XLSX, XLSM, CSV or XML) Expected Output: Success message with the uploaded file details.
- Test Case 2: Validating file upload for incorrect file type Input: Upload a file with incorrect file type (e.g., TXT) Expected Output: Error message indicating invalid file type
- Test Case 3: Validating file upload for file size limit Input: Upload a file that exceeds the maximum allowed file size of 10 MB Expected Output: Error message indicating file size limit exceeded.
- Test Case 4: Validating file upload for maximum number of columns Input: Upload a file with more than the maximum allowed number of columns (20) Expected Output: Error message indicating maximum number of columns exceeded..
- Test Case 5: Validating file upload for mandatory columns Input: Upload a file with missing mandatory columns Expected Output: Error message indicating missing mandatory columns.
- Test Case 6: Validating file upload for unique key constraint Input: Upload a file with duplicate records in the unique key column Expected Output: Error message indicating violation of unique key constraint.
- Test Case 7: Validating file upload for column length constraint Input: Upload a file with data that exceeds the maximum allowed length for a column Expected Output: Error message indicating column length exceeded.
- Test Case 8: Validating file upload for data type constraint Input: Upload a file with data that does not match the expected data type for a column Expected Output: Error message indicating invalid data type for column.

CHAPTER 4

OVERVIEW

TEST CASES:

Case 1: ✓
Case 2: ✓
Case 3: ✓
Case 4: ✓
Case 5: ✓
Case 6: ✓
Case 7: ✓
Case 8: ✓
Case 9: ✓

FUNCTIONAL TESTING LOGS

The below conditions design and determine the positive and negative testing cases of the whole application that is ready to be deployed. These are based on the numerous validation constraints that are set by the customer with a customized order of validation for optimum performance.

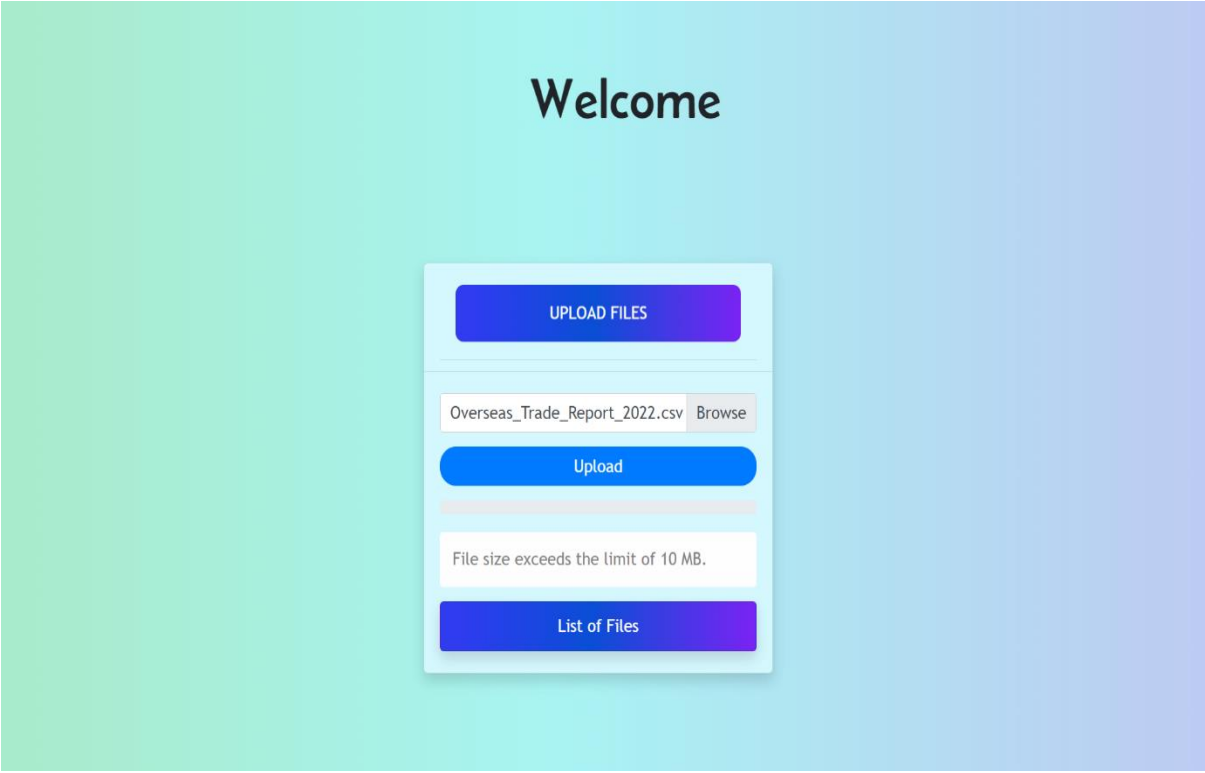
Case 1: File should not Exceed 10MB

Positive Condition:

If the file size is less than 10MB: ***Upload the file*** *if* ***all*** *the* ***other conditions*** *are* ***true.***

Negative Scenario:

If the File size should be more than 10MB : ***Throw Warning***



As observed, the file that harbors more file size than the set limit throws the warning to the user.

Case 2: The allowed file types are csv, xlsx, xlsm, and xml.

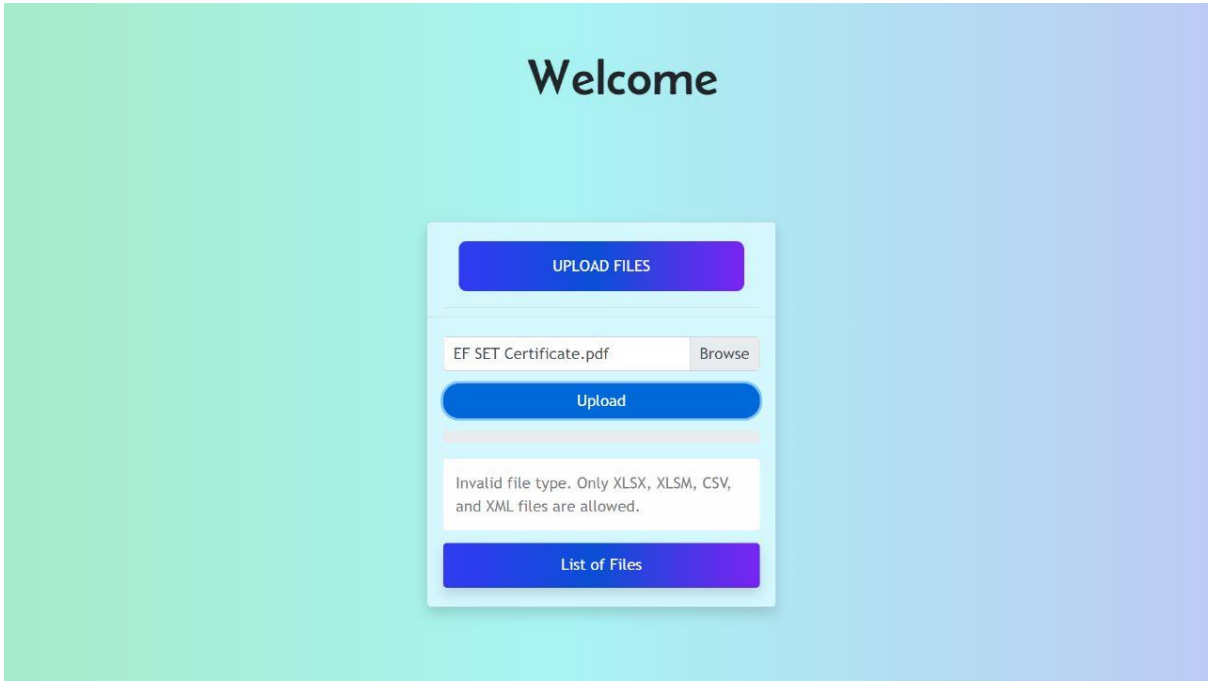
Negative Scenario:

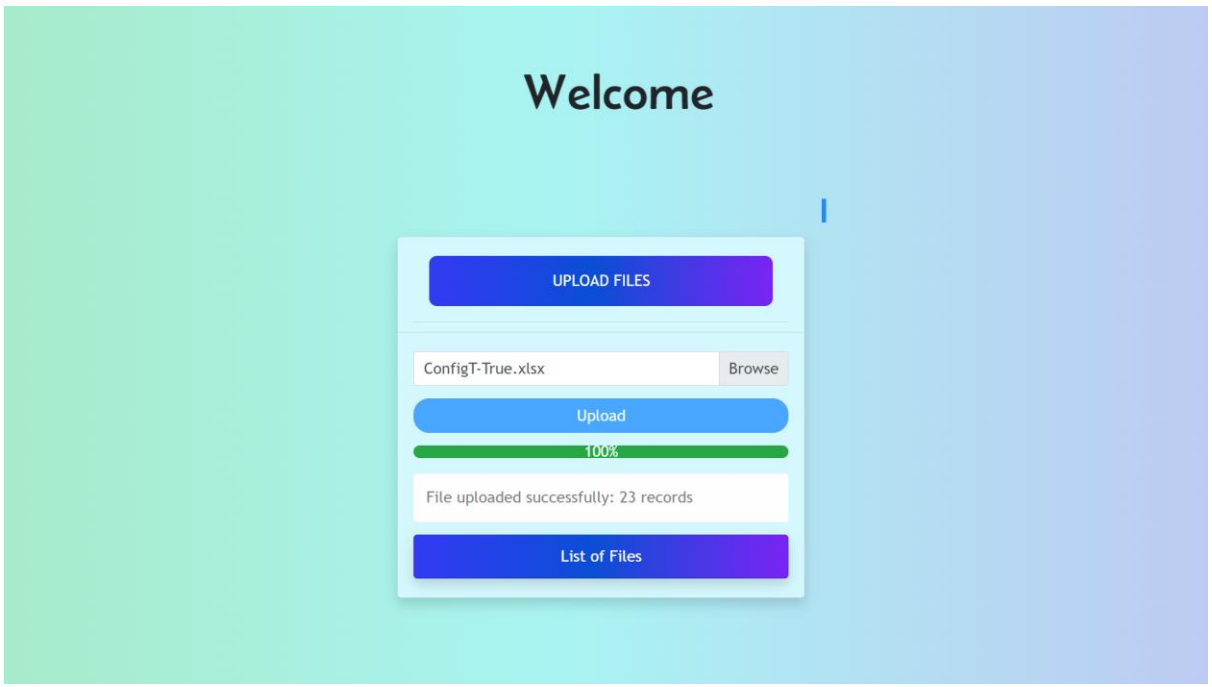
If the **file type** is not in **FTypes** then ***Show error***

Positive Scenario:

If the **file type** is in **FTypes** then **check** for all the **other conditions** and ***UPLOAD***

FTypes = [csv, xlsx, xlsm and xml]





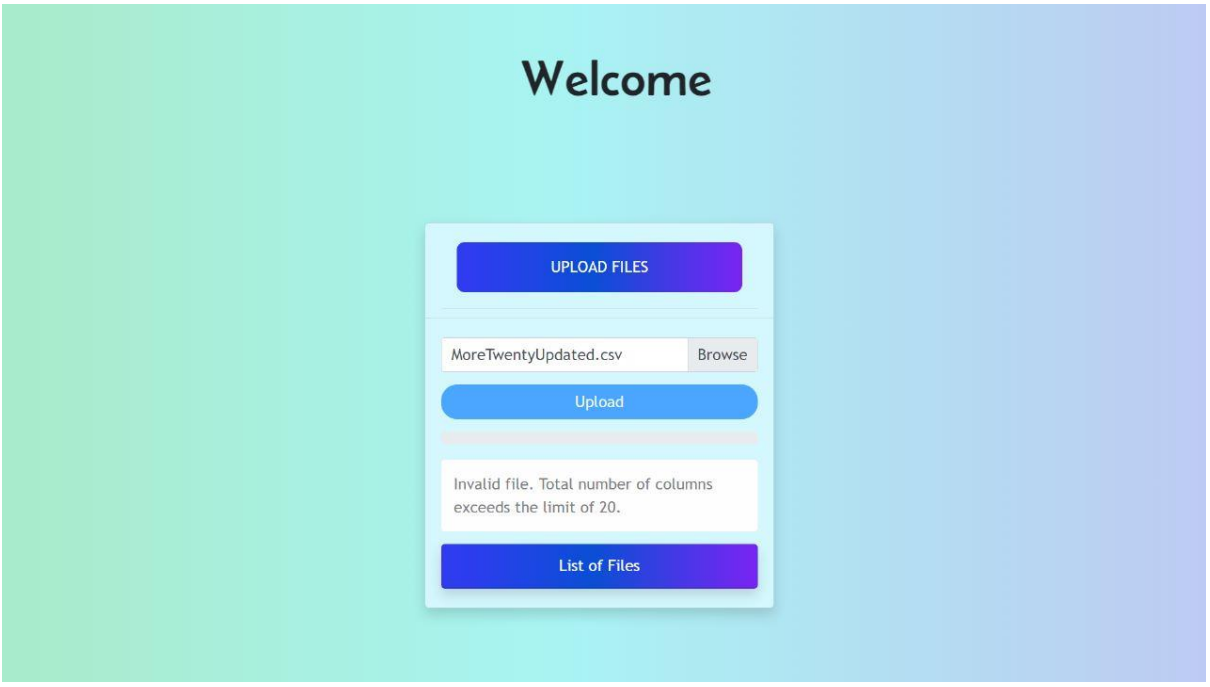
Case 3: File should support a maximum of 20 Columns

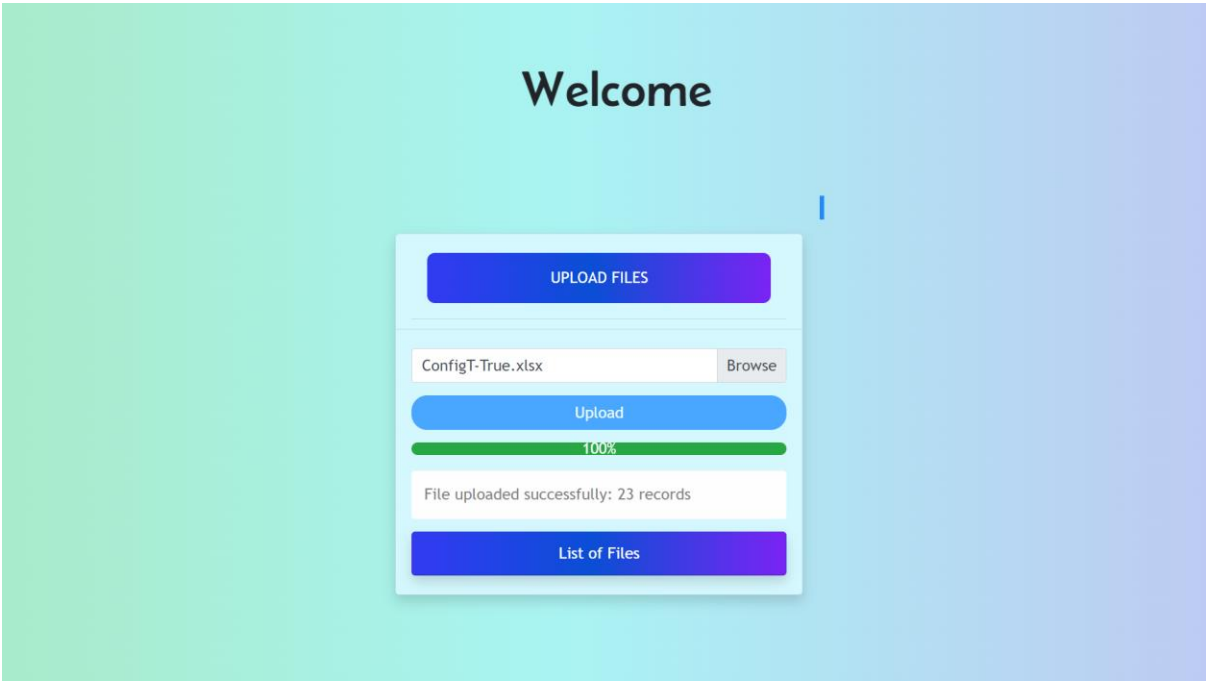
Positive Condition :

If the **number of columns** is less than 20 then **check** for all the **other conditions** and ***UPLOAD***

Negative Scenario:

If columns are **more** than **20** then ***Throw Warning to user***





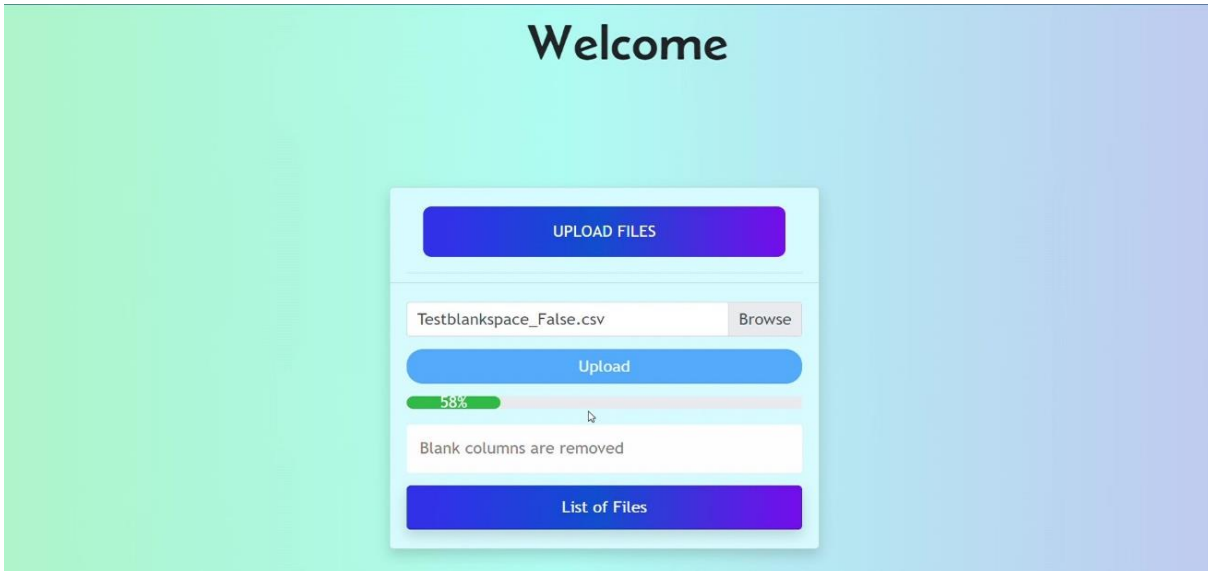
Case 4: The system should ignore and decrepitate blank columns.

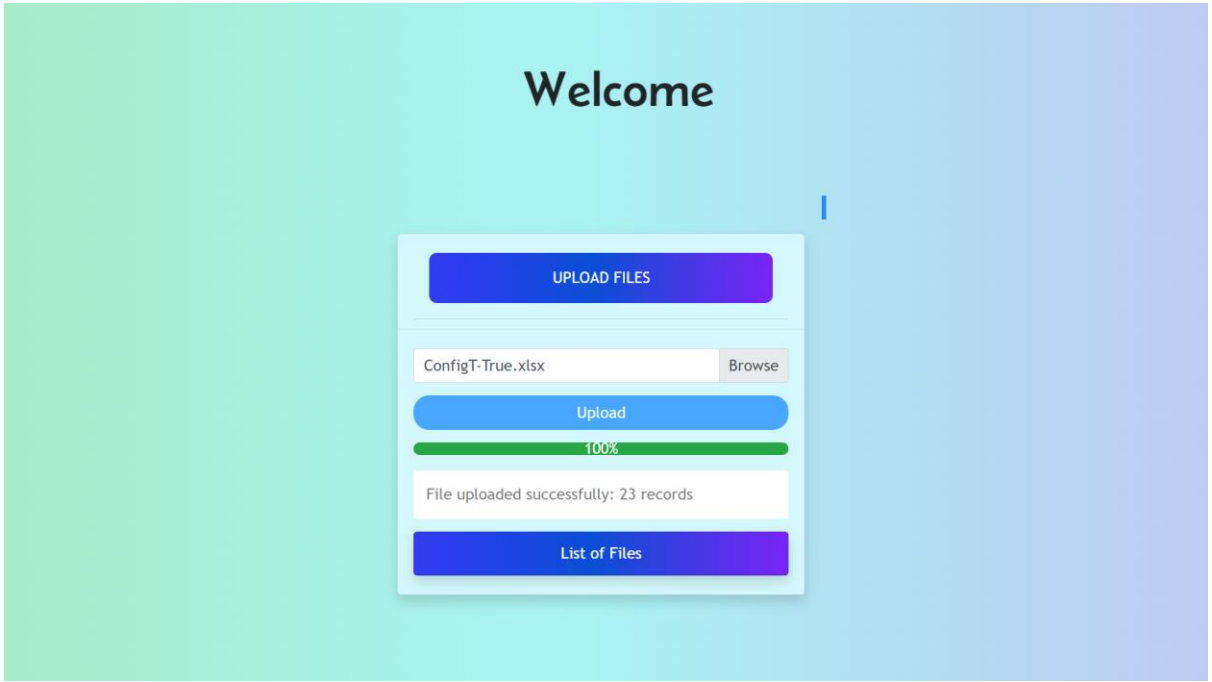
Positive Scenario:

If the does not have any blank columns then **check** for all the **other conditions** and ***UPLOAD***

Negative Scenario:

If the does not have any blank columns then **remove the blank columns, check** for all the **other conditions,** and **UPLOAD**





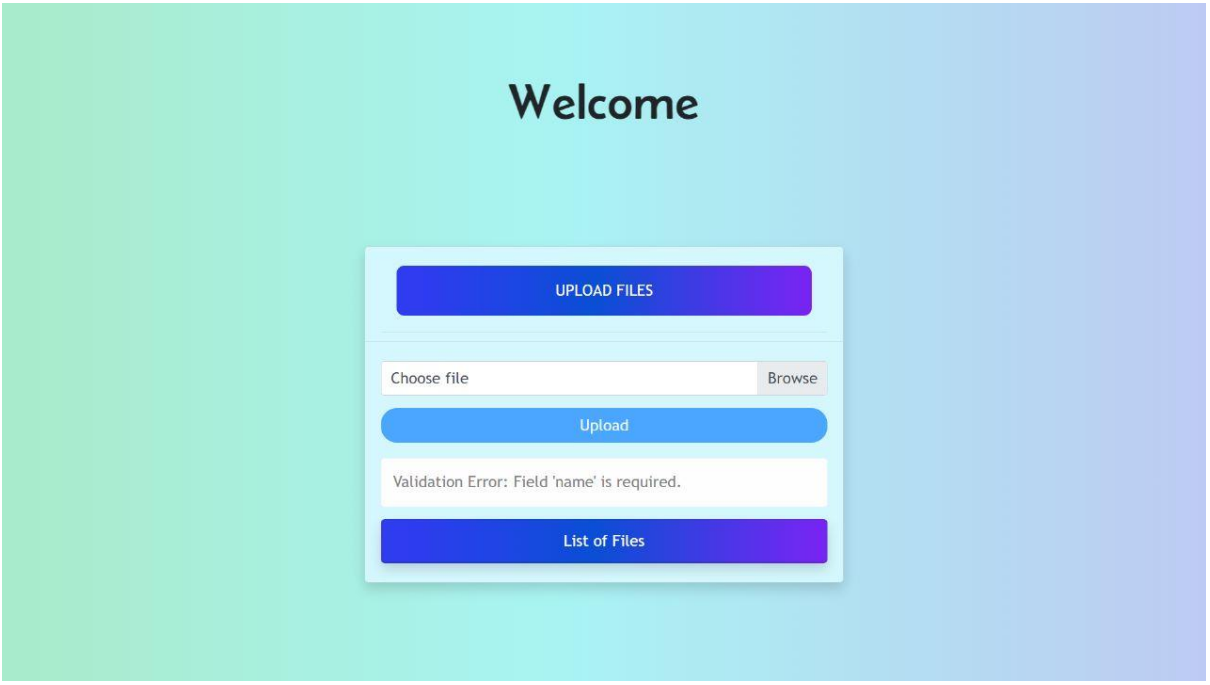
Case 5: Check for mandatory columns

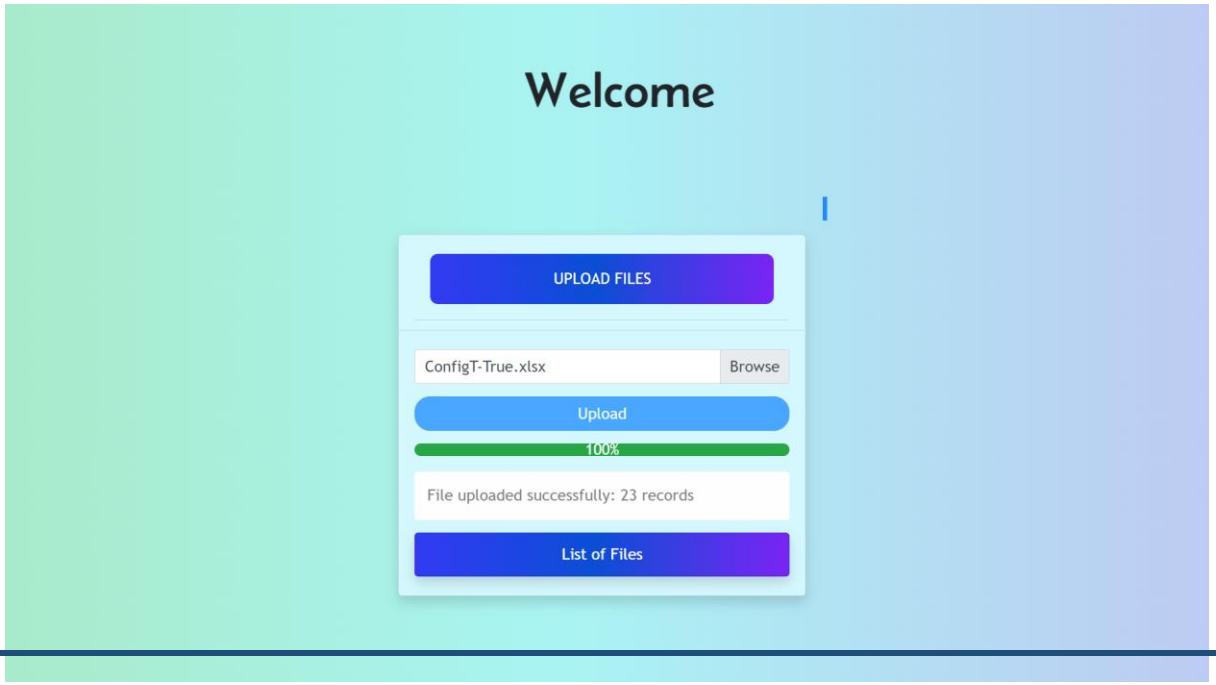
Positive Scenario:

If the uploading **file has** all the **mandatory column** then **check** for all the **other conditions** and ***UPLOAD***

Negative Scenario:

If the uploading **file does not** have all the **mandatory column** then ***Throw Warning***





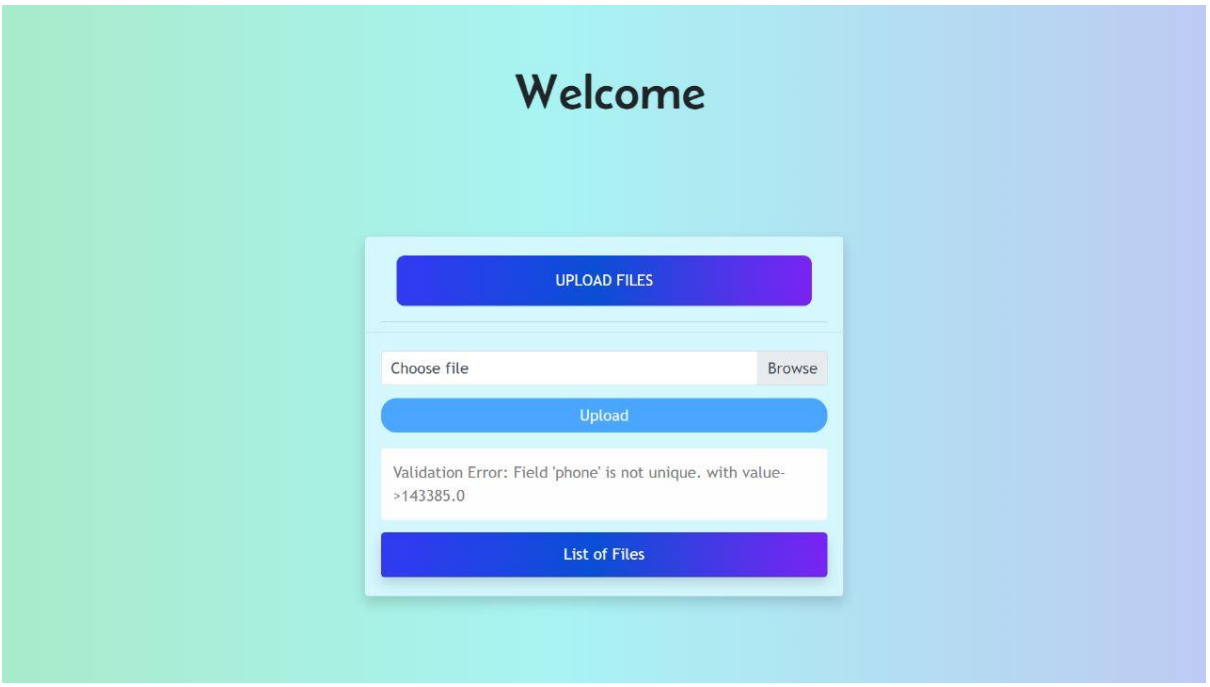
Case 6: Unique column should be unique

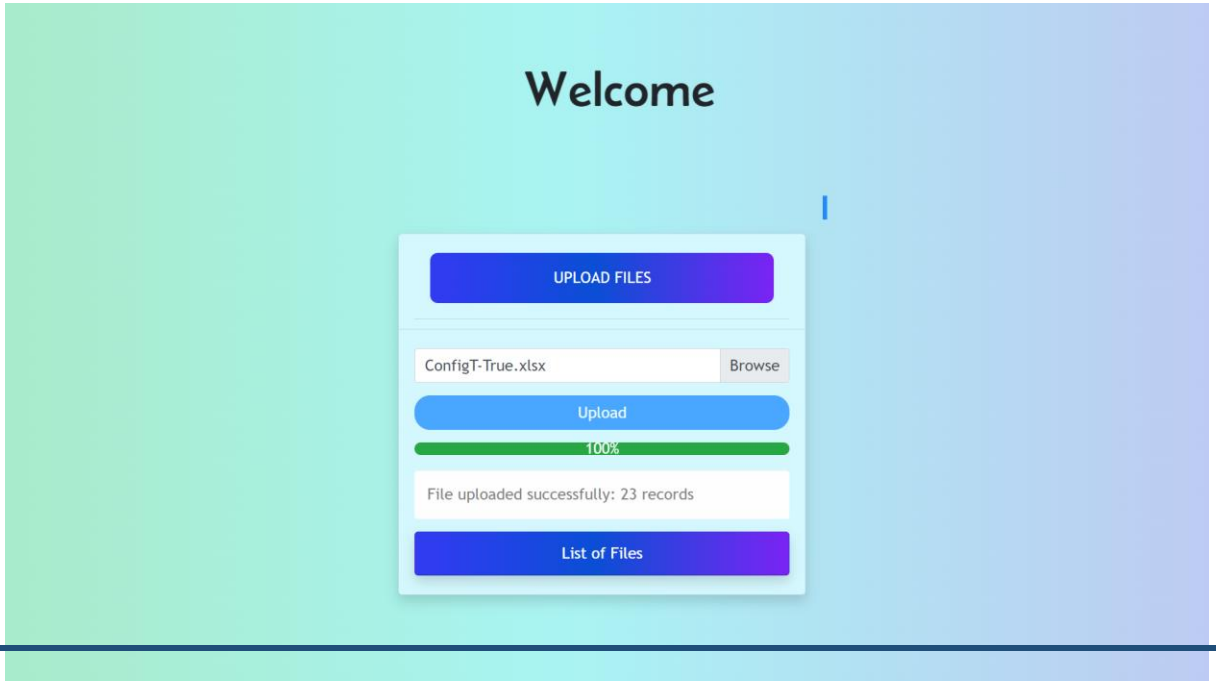
Positive Scenario:

If the file has **any unique columns** that **does not have any duplicates** then **check** for the **other conditions** and ***UPLOAD.***

Negative Scenario:

If the file has **any unique columns** that **has duplicates** then ***Throw Warning*** *and* ***Show where the duplicate exists.***





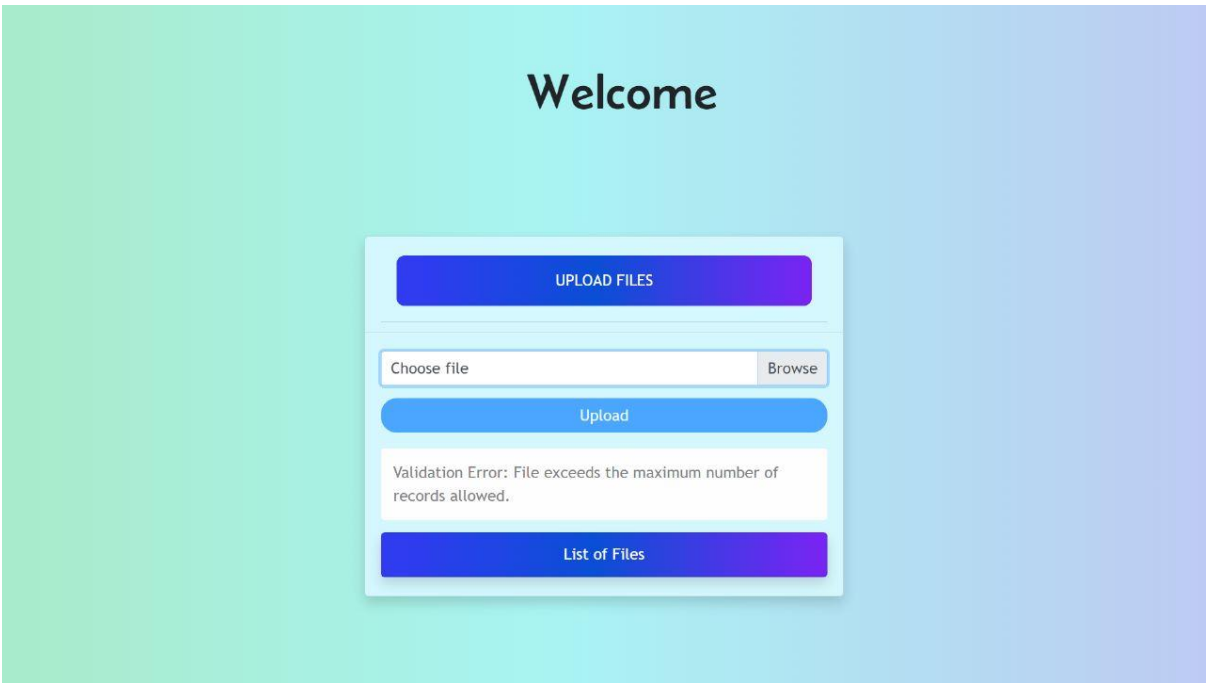
Case 7: The system should validate column length.

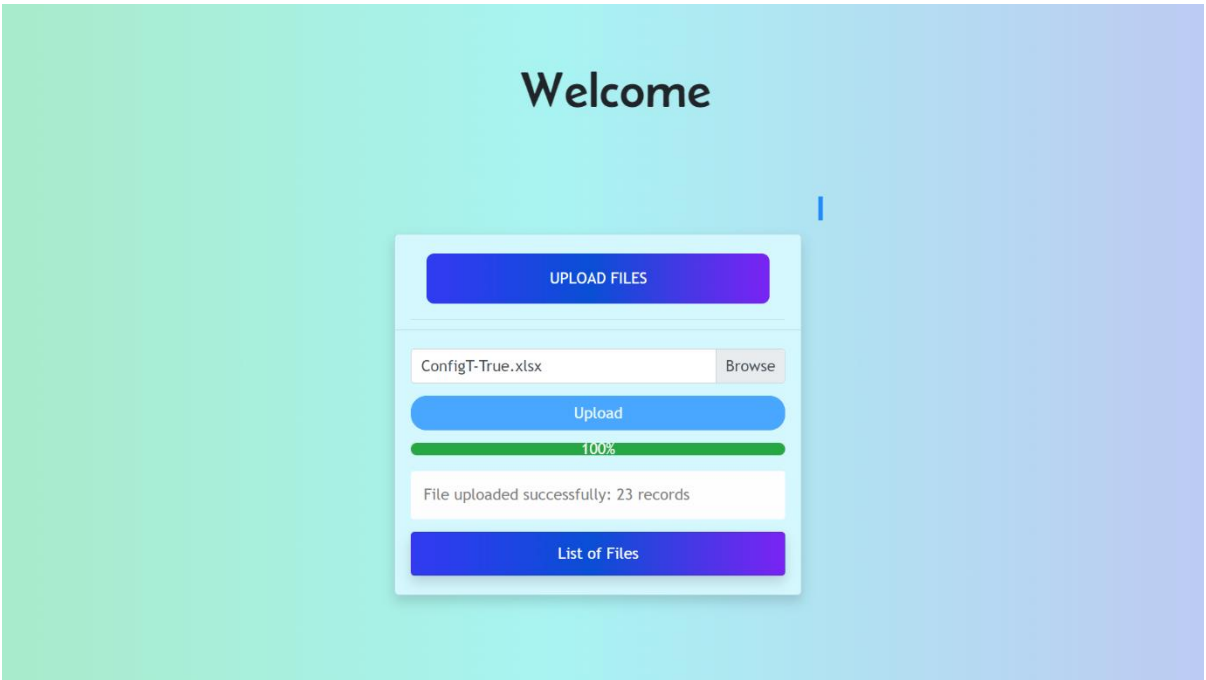
Positive Scenario:

If the **columns** have records **within a fixed column length** then **check** for all the **other conditions** and **UPLOAD**

Negative Scenario:

If the **columns** have records that exceed **a fixed column length** then ***Throw Warning.***





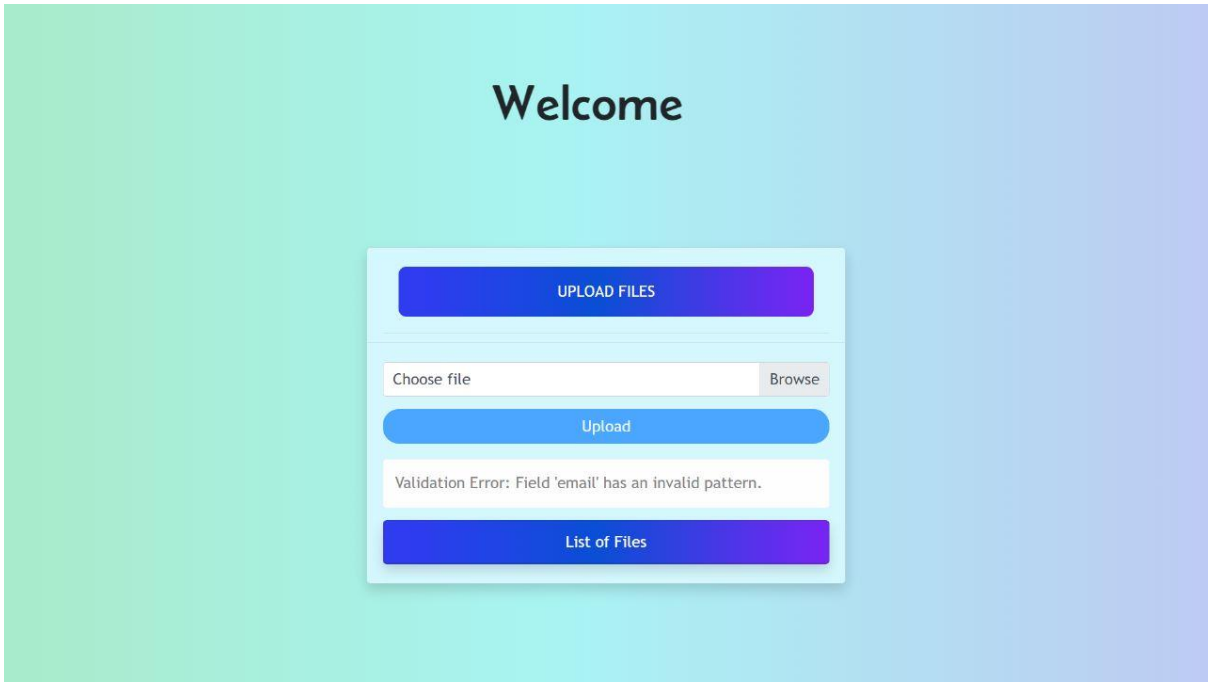
Case 8: The system should validate the data type of the column.

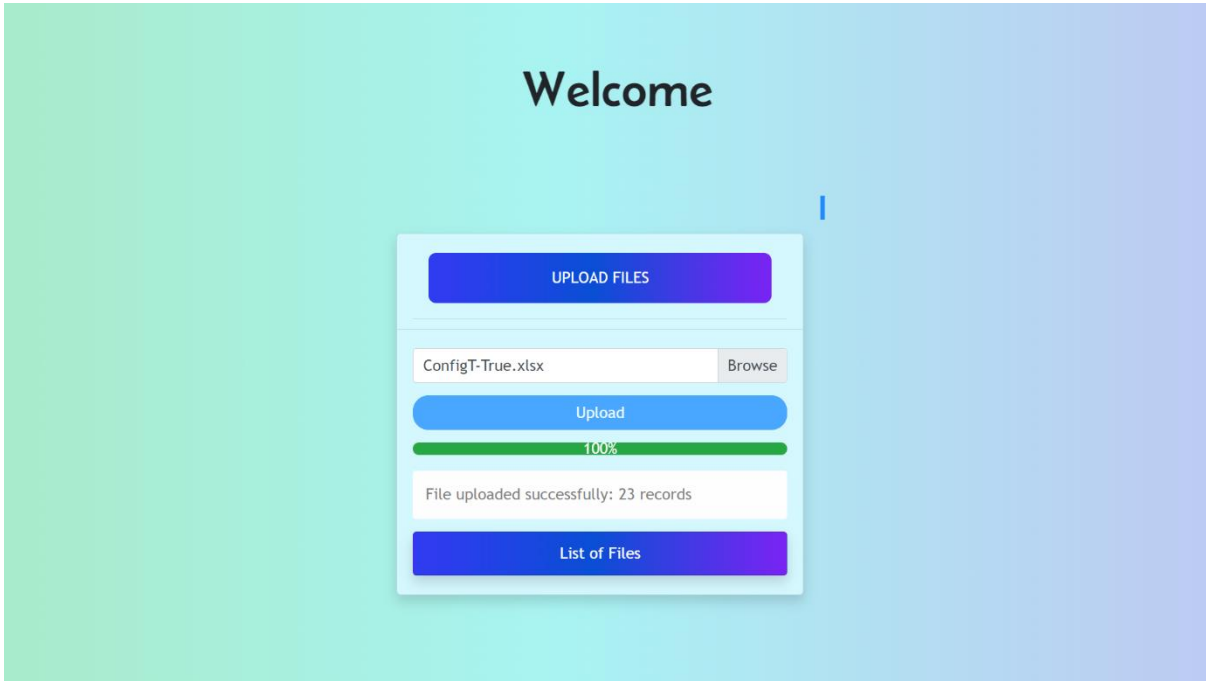
Positive Scenario:

If the column **datatype matches** with the inserted row then **check** for all the **other conditions** and ***UPLOAD***

Negative Scenario:

If the column **datatype doesn't match** with the inserted row then ***Throw Warning.***





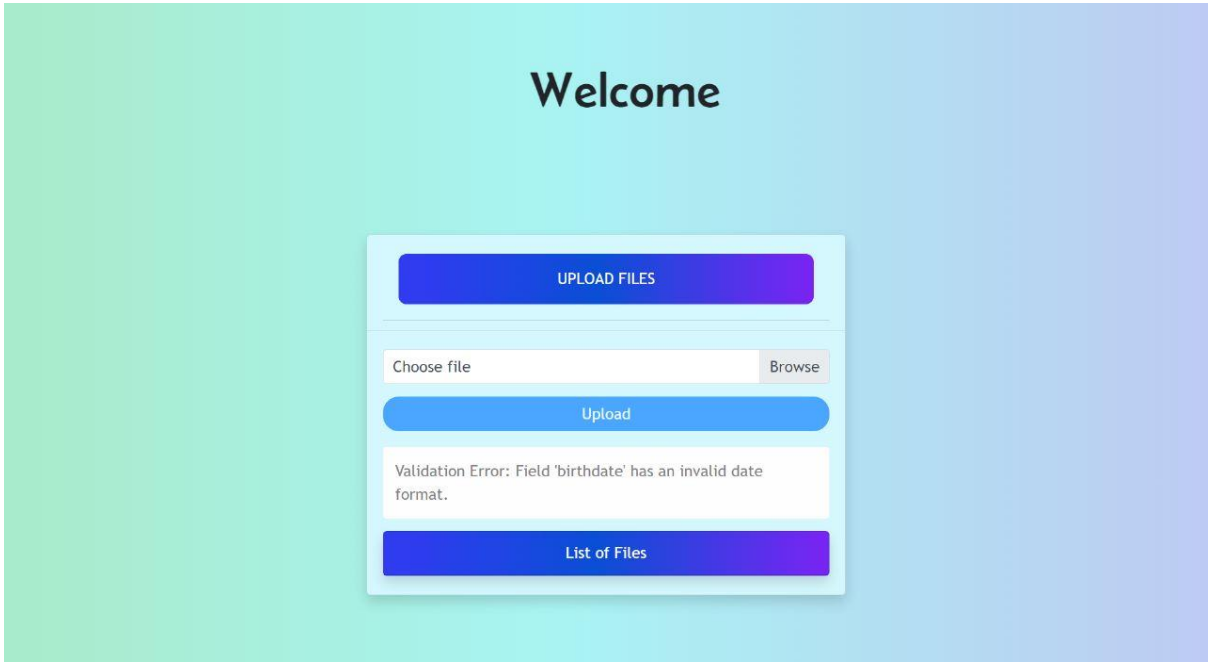
Case 9: The system should also validate date format.

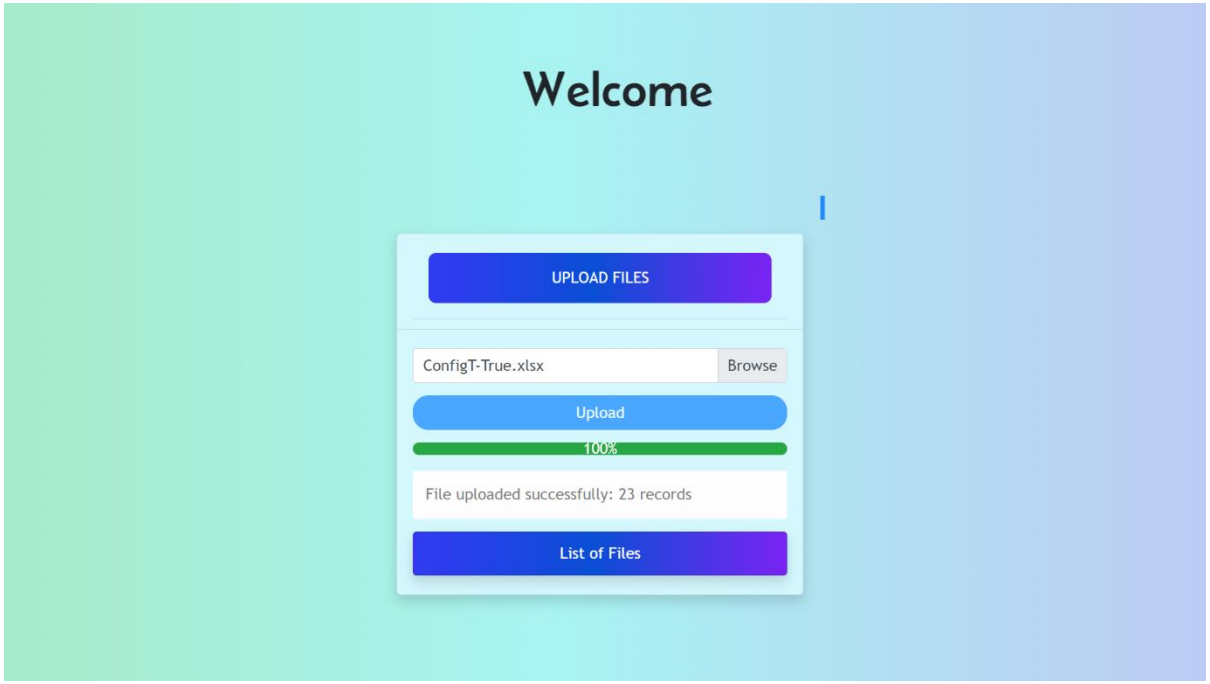
Positive Scenario:

If the column **date Format matches** with the inserted row then **check** for all the **other conditions** and ***UPLOAD***

Negative Scenario:

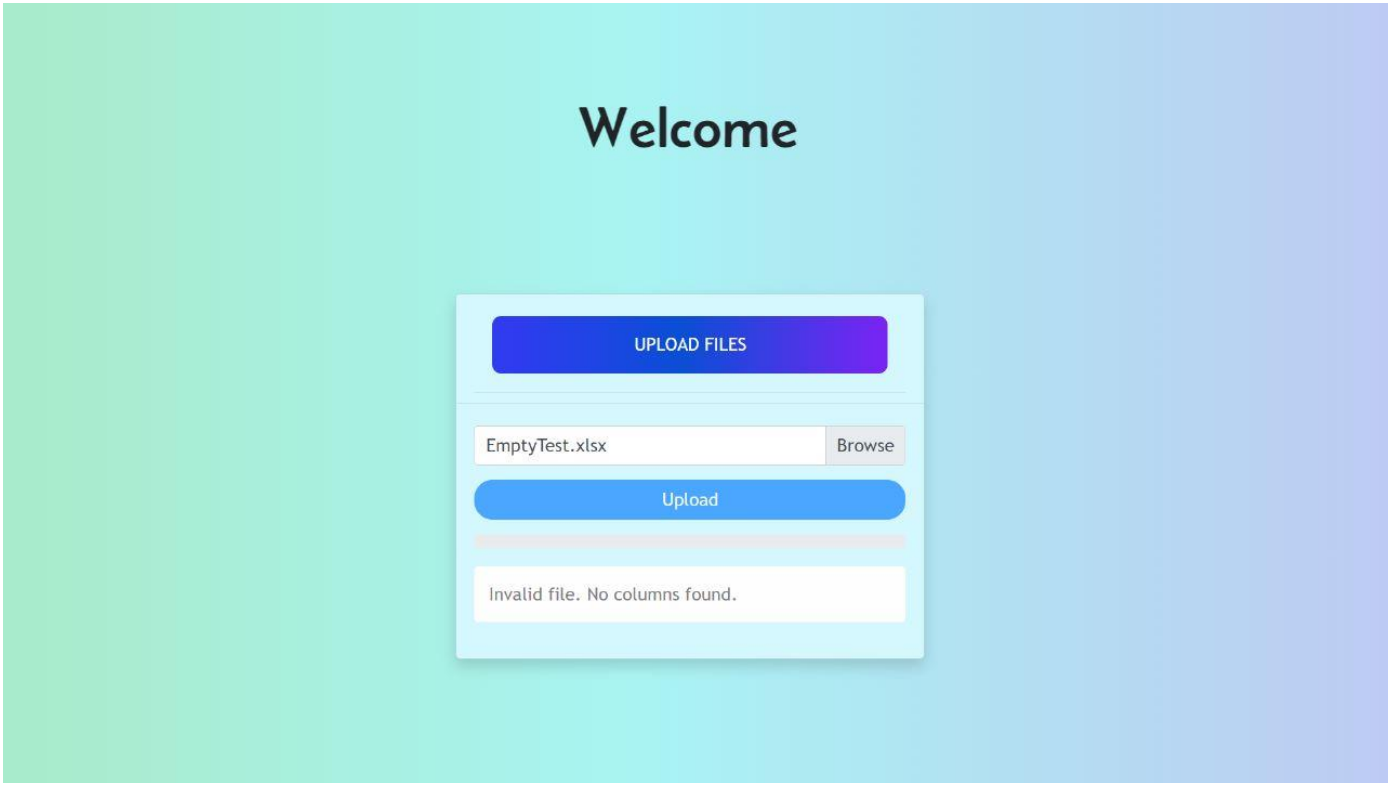
If the **date** column **format doesn't match** with the decided format then ***Throw Warning.***





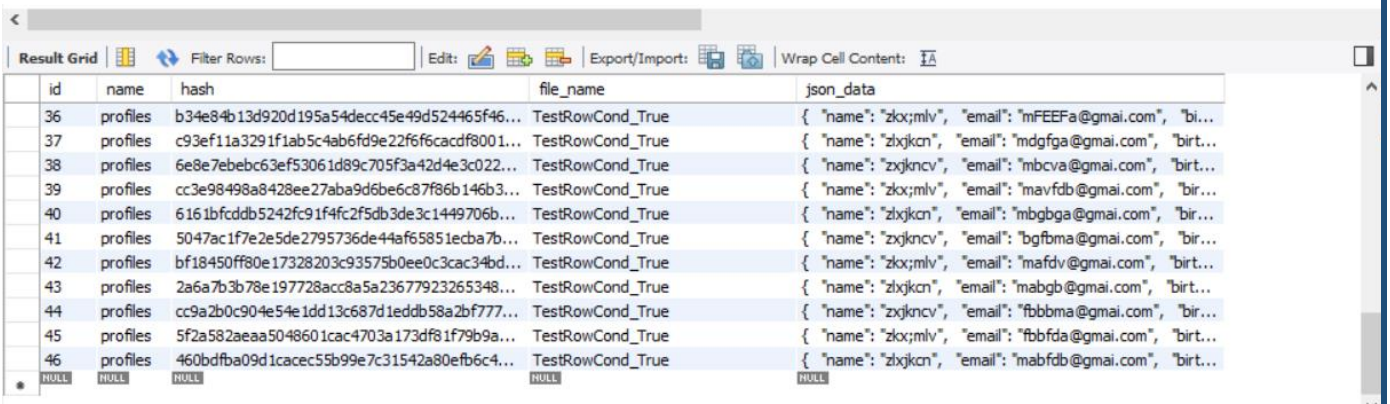
Border Case Scenarios:

- **The system should also validate a completely empty file.**
  Positive Scenario:
  If the file is not empty then **check** for all the **other conditions** and ***UPLOAD***
  Negative Scenario:
  If the **file is empty** then ***Throw Warning.***



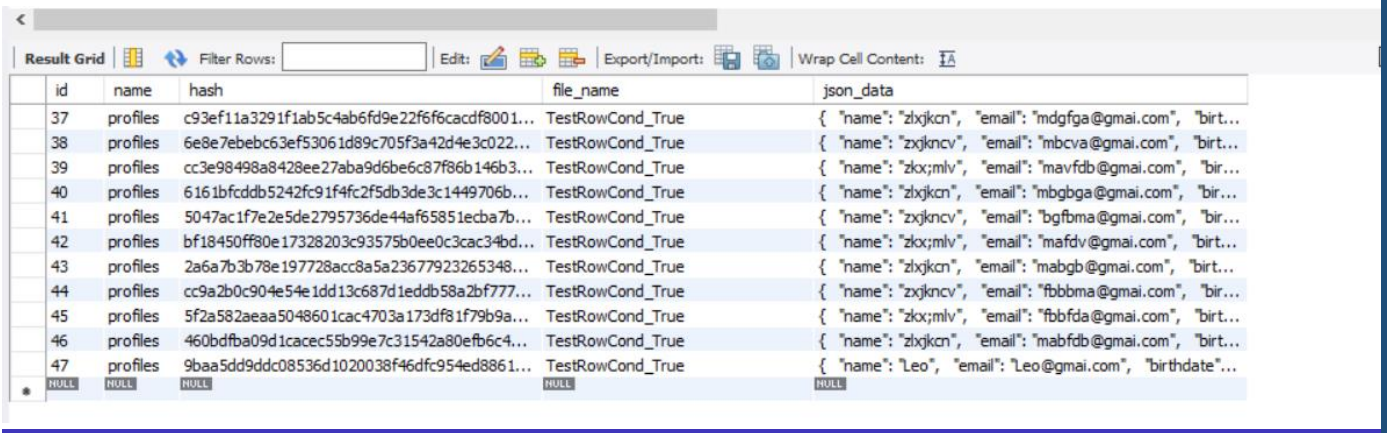
- **The System should update any changes made to the file that is uploaded again.**
  Positive Scenario:
  If the file is not having **any changes then check** for all the **other conditions** and ***UPLOAD***
  Negative Scenario:
  If the file is not having **any changes then *Throw Warning*.**

Before:



id	name	hash	file_name	json_data
36	profiles	b34e84b13d920d195a54decc45e49d524465f46...	TestRowCond_True	{ "name": "zkx;mlv", "email": "mFEEFa@gmai.com", "bi...
37	profiles	c93ef11a3291f1ab5c4ab6fd9e22f6f6cacdf8001...	TestRowCond_True	{ "name": "zlxjkcn", "email": "mdgfga@gmai.com", "birt...
38	profiles	6e8e7ebebc63ef53061d89c705f3a42d4e3c022...	TestRowCond_True	{ "name": "zxjkncv", "email": "mbcva@gmai.com", "birt...
39	profiles	cc3e98498a8428ee27aba9d6be6c87f86b146b3...	TestRowCond_True	{ "name": "zkx;mlv", "email": "mavfdb@gmai.com", "bir...
40	profiles	6161bfcddb5242fc91f4fc2f5db3de3c1449706b...	TestRowCond_True	{ "name": "zlxjkcn", "email": "mbgbga@gmai.com", "bir...
41	profiles	5047ac1f7e2e5de2795736de44af65851ecba7b...	TestRowCond_True	{ "name": "zxjkncv", "email": "bgfbma@gmai.com", "bir...
42	profiles	bf18450ff80e17328203c93575b0ee0c3cac34bd...	TestRowCond_True	{ "name": "zkx;mlv", "email": "mafdv@gmai.com", "birt...
43	profiles	2a6a7b3b78e197728acc8a5a23677923265348...	TestRowCond_True	{ "name": "zlxjkcn", "email": "mabgb@gmai.com", "birt...
44	profiles	cc9a2b0c904e54e1dd13c687d1eddb58a2bf777...	TestRowCond_True	{ "name": "zxjkncv", "email": "fbbbma@gmai.com", "bir...
45	profiles	5f2a582aeaa5048601cac4703a173df81f79b9a...	TestRowCond_True	{ "name": "zkx;mlv", "email": "fbbfda@gmai.com", "birt...
46	profiles	460bdfba09d1cacec55b99e7c31542a80efb6c4...	TestRowCond_True	{ "name": "zlxjkcn", "email": "mabfdb@gmai.com", "birt...

After:



id	name	hash	file_name	json_data
37	profiles	c93ef11a3291f1ab5c4ab6fd9e22f6f6cacdf8001...	TestRowCond_True	{ "name": "zlxjkcn", "email": "mdgfga@gmai.com", "birt...
38	profiles	6e8e7ebebc63ef53061d89c705f3a42d4e3c022...	TestRowCond_True	{ "name": "zxjkncv", "email": "mbcva@gmai.com", "birt...
39	profiles	cc3e98498a8428ee27aba9d6be6c87f86b146b3...	TestRowCond_True	{ "name": "zkx;mlv", "email": "mavfdb@gmai.com", "bir...
40	profiles	6161bfcddb5242fc91f4fc2f5db3de3c1449706b...	TestRowCond_True	{ "name": "zlxjkcn", "email": "mbgbga@gmai.com", "bir...
41	profiles	5047ac1f7e2e5de2795736de44af65851ecba7b...	TestRowCond_True	{ "name": "zxjkncv", "email": "bgfbma@gmai.com", "bir...
42	profiles	bf18450ff80e17328203c93575b0ee0c3cac34bd...	TestRowCond_True	{ "name": "zkx;mlv", "email": "mafdv@gmai.com", "birt...
43	profiles	2a6a7b3b78e197728acc8a5a23677923265348...	TestRowCond_True	{ "name": "zlxjkcn", "email": "mabgb@gmai.com", "birt...
44	profiles	cc9a2b0c904e54e1dd13c687d1eddb58a2bf777...	TestRowCond_True	{ "name": "zxjkncv", "email": "fbbbma@gmai.com", "bir...
45	profiles	5f2a582aeaa5048601cac4703a173df81f79b9a...	TestRowCond_True	{ "name": "zkx;mlv", "email": "fbbfda@gmai.com", "birt...
46	profiles	460bdfba09d1cacec55b99e7c31542a80efb6c4...	TestRowCond_True	{ "name": "zlxjkcn", "email": "mabfdb@gmai.com", "birt...
47	profiles	9baa5dd9ddc08536d1020038f46dfc954ed8861...	TestRowCond_True	{ "name": "Leo", "email": "Leo@gmai.com", "birthdate"...

Used File: TestRowCond_True.csv

TEST LOGS

ERROR LOGS

Error:	**#1**
Reason :	Failed to determine a suitable driver class.
Target Location :	FileDataRepository.class
Description:	Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.
Reason:	Failed to determine a suitable driver class
Action:	If you want an embedded database (H2, HSQL or Derby), please put it on the classpath. If you have database settings to be loaded from a particular profile you may need to activate it (no profiles are currently active).
Type of ERROR:	Appliction Failed To Start.

Error :	**#2**
File Type :	Repository.FileDataRepository' in your configuration.
Target Location :	Consider defining a bean of type 'com.files.upload.
Description :	Parameter 0 of constructor in com.files.upload.service.FileDataService required a bean of type 'com.files.upload.Repository.FileDataRepository' that could not be found.The injection point has the following annotations-@org.springframework.beans.factory.annotation.Autowired(required=true)
Action:	Consider defining a bean of type 'com.files.upload.Repository.FileDataRepository' in your configuration.
Type of Error :	@org.springframework.beans.factory.annotation.Autowired (required=true).

Error : **#3**
Error Type : No 'Access-Control-Allow-Origin.
Description: Access to XMLHttpRequest at 'http://localhost:8080/upload/profiles?fileName=varaprasad' from origin 'http://localhost:4200' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.Access to XMLHttpRequest at 'http://localhost:8080/upload/profiles?fileName=varaprasad' from origin 'http://localhost:4200' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin'
File Type : XML Type

Error : #4
Error Type : **Sonar Cloud**
Description: To see the full stack trace of the errors, re-run Maven with the -e switch.Re-run Maven using the -X switch to enable full debug logging.For more information about the errors and possible solutions.
File Type: JDK 11

Error: #**5**
Error Type: 200~https' is not supported
Description: In the working copy of 'Desktop/SpringApplication/README.md', LF will be replaced by CRLF the next time Git touches it.'Desktop/SpringApplication/.mvn/wrapper/MavenWrapperDownloader.java', LF will be replaced by CRLF the next time Git touches it.
File Type: Spring Appliction

==**==========**