FS Import Converter

Project Report

Chelsea Case-Miller

Sandgate Services Ltd

Contents

Introduction

In this report, I will showcase my thought process and actions when planning, creating, troubleshooting and publishing this programming project.

The need for the project, named FS Import Converter, had arisen due to inefficiencies in the process that the IT team was using to import client disbursement data. The previously used process involved manually executing SQL scripts on data exported from third-party wealth management providers, which were likely to be in an xlsx format, with variance in what columns the data was located in and how many columns the data included. The manual process was not only time-consuming but was also prone to errors and inconsistencies due to the frequent changes in the input files. These issues have led to data inaccuracies, delays, and increased workload for the IT team, which could be making better use of time on their higher-priority workloads.

To address these challenges, the solution is a web application designed to automate the data import process as much as possible. The application has a user-friendly interface that simplifies the data ingestion process, allowing users to upload CSV or Excel files, attempt to automatically map input columns to output columns, perform database lookups, and generate a properly formatted CSV file to then be imported to the CRM. By including CI/CD practices this guarantees a smooth development and deployment pipeline.

“As the line manager of the apprentice Chelsea Case-Miller, I Neil Forrest confirm that the project report for the FS Import Converter is both valid and attributable to the apprentice and the project.”

Project Scope

Need: Sandgate Services IT requires a web application to streamline the import and handling process of FS Import data from the wealth management team.

Scope: The Sandgate Services Development team will create a web app that includes functionality to ensure fast, easy and repeatable improvements to the FS Import procedure.

Deliverables:

- User interface accessible via a browser.
- Multiple upload file types handling (XLSX, XLS and CSV).
- Automatic and manual column mapping.
- Database integration to look up additional client data.
- Data export to a CSV file type.

Exclusions:

- Data sanitation and checking of correctness of data provided in the input file.
- Any write or delete operations on the CRM database.

Constraints: The project must be completed within a 9-week timeline, and must comply with both organisational policies and legal requirements such as internal security practices and GDPR (General Data Protection Regulation).

Assumptions: Any graphics such as logos should already be created by the client. There also should be an appropriate IIS web server set up ready for the web application to be published to this, this does not include individual site creation as this will be handled by the development team.

Legal & Organisational Requirements

Legal Requirements:

To ensure that the program is compliant with the law, planning stage meetings were held to discuss legal requirements and what this meant regarding the project. This was to ensure the program was compliant with current legislation.

The main legal requirement that was identified was GDPR (General Data Protection Regulation). In short, GDPR is the regulations surrounding how businesses should legally collect, handle, store and interact with personal data. The program will be handling amounts relating to transfers to/from account balances concerning the wealth department procedures, this means the program will have to ensure data is not stored insecurely and that it is inaccessible to any persons that may wish to view this sensitive information criminally.

The other requirement that was identified was the requirement to ensure any third-party tools or libraries used should be used in compliance with software licencing and usage rights.

Organisational Policies:

[B3] The identified organisational policies that would impact the program are the following; smart card login and 2FA (Two Factor Authentication). This is an organisational policy that ensures account security by requiring users to use a smart card to physically log into their work computers, this adds a large amount of security to our internal systems, ensuring that only those who are allowed access are granted it. Alongside smart card logins, all online accounts and tools we use that can have 2FA enabled should be set up to be used with our YubiKey FIDO authentication devices. Once again ensuring only authorised users can log in and interact with code and other environments.

The other organisational policy identified was to ensure that all pushed code is approved and tested before being pushed to production and the main branch. This ensures that all code uploaded gets a second user check ensuring best practices are being followed and any issues are caught before making its way to the main code base. The tests also ensure that the code produced works in conjunction with existing functions ensuring that pushed code does not have any adverse effects to the already working program.

Analysis & Problem Solving Within the Project Lifecycle

During the project, I faced several challenges in which I had to use my knowledge of debugging, research and programming to overcome them.

One of the main challenges I faced was that of Excel files being stored not in plain text, but in Microsoft's file standard. This meant I had to find a way to read the contents of the file whether that be deconstructing the file structure and making my own parser, or what I ultimately chose in the end product, find and use an already established package that can do this for me. In the end, SpireXLS, the NuGet package I opted to use, has been stable and easy to work with in relation to the wider project and has saved a lot of time programming compared to a custom solution for the problem.

[S7] Whilst developing the function that handles file uploads and then sends to my ExcelToCSV function, I came across an error when trying to use the file stream from the upload (see fig. 5) where I had found that an uploaded file, although a file stream, is an Http.ReferenceReadStream and is unable to be converted to a standard FileStream. This is due to the limitations of these object types. After doing some research on different Streams and how files could be handled, I came up with a simple solution. This was to save the uploaded stream to a temp file, then send the file path to the ExcelToCSV function (see fig. 10). I also changed how the ExcelToCSV function loaded the spreadsheet by using the LoadFromFile method rather than the LoadFromFileStream method. This was a simple solution to what I had originally thought was a complex issue that would halt the development process.

Research & Findings

[B2] The significant research I had done surrounding the project was to do with the Excel file format and how I could convert this into a programmatically easy-to-read object within the program. My findings about Excel files were that they are stored in a file format that by default cannot be read with code such as `with open("filename.xlxs") as file` this is due to the xlsx file type being a type of zip file, which contains multiple files within that relate to the content, styling, functions and sheets. This is based on the SpreadsheetML file type which uses xml files to determine the content mentioned above. Although I could have implemented a tool to unzip the files and parse the Excel document, my research brought me across a pre-made NuGet package that would save a lot of time in development, which was crucial to the overall project.

Project Outcomes

As a result of releasing the FS Import converter, the IT manager has reported much fewer time entries surrounding this task by members of the IT department, therefore allowing them to focus on their primary job roles as first-line support and client IT requirements.

Alongside the outcomes that the project directly relates to, in terms of business use. I also learned and reinforced my understanding of the technologies used within the software development team. This included my in-depth research on SpreadsheetML formats, and NuGet packages as well as reinforcing my knowledge of ASP.NET core (MVC) written specifically in the C# language.

Recommendations and Conclusions

Now that the program has been released and used by the members of the IT team, the next step is to consider improvements and recommendations that could be made for future releases.

The first improvement would be to improve the auto column mapping, as sometimes this can misidentify columns which, although the users can manually select the columns, slows down the task that the program has been developed to quicken. My recommendation is to both look at improving the column header search and also include a data type lookup for each column to best identify amounts which would show as floats, client names as strings etc. Both of these improvements would quicken the time that users need to spend manually mapping columns and would ensure that columns are mapped automatically. In the best case scenario providing that the auto mapping is accurate enough, the manual mapping feature could be removed making the program only require 3 inputs by the user instead of the current 4.

I would also recommend that the project is still best suited to be developed as an ASP.NET Core (MVC) program, this is due to the robustness and scaling of the language making any future development quick and easy to integrate with the existing environment. Continuing to act as a web app also ensures that the requirements for end users are just a web browser and therefore does not need any additional software to be installed to ensure that the task can be completed.

The final program can be seen in figures 21, 22, 23 and 24

SDLC Within the Project

Planning

During the planning stage, I had multiple meetings with the IT manager to discuss how the project would progress and what features were required, and where these would fit into the overall program flow and project lifecycle.

[K2] During this stage, we discussed how the project would be managed and carried out to meet the requirements of the scope, this included who would be responsible for what portions of the project, in this case, the IT manager would be acting as the project lead, ensuring that any requirements are well discussed and planned to ensure that the project meets deadlines. As for myself, I would be acting as the lead, and also the sole developer. This is due to other members of our team being focused on other projects and first-line support via our helpdesk. As the lead developer, my responsibilities would be creating the majority of the code base, ensuring that tests are written and passed alongside ensuring code is written to a high standard along with code comments to ensure any future development will go smoothly. Again due to limitations to staffing, it was agreed that I would also carry out the responsibilities of testing the program and writing any CI/CD workflows that are required.

Analysis

[K11] In our project analysis meeting the IT manager and I took a look at how the current process was undertaken by running through a mock import, this gave us a good starting point to get an idea of what features were required in the program and where improvements could be made.

Once a good understanding of the requirements and purpose of the program, we wrote up an outline document for the project requirements, which included the content below:

Purpose & Goal:

The project that has been collectively named "FS Import Converter" will aim to solve the following problems with the current method of disbursements and the process that goes with the process.

The current procedure is to obtain Excel documents from third-party wealth management providers, in which the structure of the data often changes from month to month. Then this will get sent to the IT department to add additional client details from our client database, this is done by a member of IT writing a SQL query to pull additional data, to add to the Excel report, which is a very time-consuming part of the process. Finally, the data is then imported into our client database so that disbursements of each client are allocated appropriately.

The main identified problems, which the program aims to solve, in this process are as follows:

1. Structure of the input file often changes meaning the merge of additional data currently has to be done manually.

2. The process takes a lot of time for the IT team who has to carry out writing SQL and merging this data, into an appropriate format for the client database.

3. In the event IT is dealing with a backlog, this process sometimes gets left longer than expected, due to the fact that only IT can apply the data and subsequent import.

There will be two main users of the program, those being the wealth management team and the IT team, the program aims to be easily used by both departments meaning the reliance on solely the IT team will be reduced.

The project is important to the company as it both solves the issue of the length of time used up in the process and will also ensure that the data is consistent throughout even when the input document is subject to formatting changes.

Features:

The main features identified for this project are as follows:

1. Have an easy-to-use and intuitive UI
2. Be able to handle both .xlsx and .csv input file types
3. Automatic and manual column mapping
4. Additional client data gathered from onsite SQL database
5. Standardised output file that can be imported back into the onsite client database

System Requirements:

The program should be written in ASP.NET core (MVC) therefore being able to run on the organisation's IIS server, allowing internal use via any web browser.

Any data connections should be carried out using SQL queries to the organisation's SQL server, whilst ensuring active connections are only open for as long as required during transactions.

Release Criteria:

Before releasing the program in any version, the program must align with the following standards.

1. Test coverage should be at least 80%
2. All unit tests must have passed.
3. Code comments should be up-to-date and detailed when the code is not intuitive or when code blocks and functions are complex.

Timeline:

The project has been agreed to be set within the timeline of 9 weeks, meaning a planned release date of the 25th of September 2024. When discussing version releases in-depth, timelines for each are to be set individually and determined by the complexity of the requirements within the version. The main project features and requirements are to be completed within the overall project timeline of 9 weeks.

Design

[K11] The design phase included coming up with ideas on how the program flow and UI (User Interface) would look. One of the main things discussed in regards to the look and feel of the program was to keep it minimal, yet also provide the users with checkpoint-like steps in the process that users can take to ensure the upload, processing and end file is what they are expecting. This should help reduce any errors that may be in either the input file or the users' use of the program. This is important especially since the program relates to funds of clients which needs to be correct when the file is uploaded into our CRM database. To achieve this minimal look we decided to keep things simple and use the in-built Bootstrap 5 CSS (Cascading Style Sheets) classes to create a simple yet effective layout, we had identified that we would need to make use of buttons, file inputs, dropdowns, tables and modals.

We also created a program flow chart (see fig. 1) to ensure we had a good understanding of how the program would work. To ensure that we had met all the requirements we ensured that we kept referring back to the analysis of the current process and the identified requirements that we had inferred from the analysis stage.

Implementation

The implementation of the design into a fully-fledged program, although time-consuming, was relatively simple since we had broken down the programs flow into an easy-to-understand flowchart. First, the GitHub repository had to be created (see fig. 2) to ensure that proper collaboration and CI/CD could be implemented. This included the repository settings such as its name, privacy and .gitignore to ensure that no user environment items are pushed to the repository. (See fig. 3) Next, I created the licence, security and readme files to ensure that others who may work on this repository understand both the program and its procedures when it comes to developing it.

[S12] After the repository was all set up, I switched my focus to actively developing the program. Looking at the flowchart I had identified that the first portion would be to create the file upload function, this would accept an upload from a user, then would check what kind of file extension it had to then either display an error, send to a convert function that would be created later, or to save as a CSV.

[S1] This function needs to pass the relevant outputs to other functions to either convert or save the file, this was the next logical step to move on to programming. I chose to next create a function to save the file in the temp directory (see fig. 11), this is going to be my main place to store files as it is both locked down to system accounts on the application server and is emptied regularly based on computer policies that are applied to our servers. This helps make sure that the information passed to the program is not held for longer than needed ensuring that a breach of GDPR is highly unlikely unless the server was compromised, in which we have many layers of security to prevent this from happening.

Following this I now needed to create a function to convert an XLSX or XLS file to csv so that I could properly loop through and extract the data. For this, I researched how an Excel file is stored. I concluded that it is not stored in plain text or any easily read format without the use of Excel or other Microsoft DLLs or services. I had researched several third-party tools that offered to convert Excel to CSV and found a reputable free NuGet library called spireXLS, this allows me to load an Excel file from a data stream and then save the file as a CSV format to be used later. (See fig. 12)

[K6] Next I had identified I had programmed what I had identified as a major feature, this now needed to be pushed to the GitHub repository to ensure that version control and collaboration is kept up to date a smoothly running without one person having a lot of code to upload relating to many different areas of the project. Pushing this to GitHub was straightforward as I had used GitHub Desktop, an app that connects to your repositories and allows you to commit changes without having to use command line git commands. You can see that after committing my changes, I asked a colleague to review and approve my changes, at which point it then got merged with the main branch. (See fig. 13)

[K9][S16][S11] Now I needed to start working on the extraction of the data within the CSV, since using ASP.NET Core (MVC) I had decided to create a model which is an object definition to store multiple types of data. Using this kind of object orientated design enabled me to write methods within the class definition. Some of these included a constructor which takes in the CSV file path, reads the data, maps the columns and saves this all to the model. (See fig. 14) The other class methods were those to save data to dynamic lists. (See fig. 15) This is due to the fact that, when looping through the data, I need to keep track of data that has been processed already and what has yet to be processed. This ensures that there are no duplicate entries that would then cause errors when importing this back into the CRM database.

Now that the data can be loaded into the model and be passed around the input and output lists whilst keeping track of what has been processed. I had opted to create the auto column mapping function. (See fig. 16) This function was designed to accept an array of strings, then search each string for pre-specified search terms. Then on a match of one of those terms, add the mapping to a list which later gets returned and saved in the model for future use.

The program is now able to use these automatically mapped columns to know what each part of data is. This is now used in a function that connects to the CRM database and runs a query that had been written for the past manual work that was done. This query (see fig. 17) returns one of four values, these being:

- 0 which means that the job name is non-existent or suspended for the given client ref.
- -1 which means the client is suspended.
- -2 which means that the client ref does not exist.
- Any other integer is the ID of the job for the given client ref.

If the value is 0,-1 or -2 then that line of data is considered an error that needs to be displayed to the user. So that they can either ignore those lines, re-import the file or assign those values to our internal client ref in a process that is called ‘bucketing’.

Finally the program will ask the user to download the formatted file which is in the end CSV format that is then importable into the CRM database. It does this using a class method to cast the output file to bytes which then the browser offers to download to the user. (See fig. 18)

Test

Testing the program was carried out using a mix of user acceptance testing and unit testing. The unit tests were written alongside each function to ensure that, when code was pushed to the GitHub repository, it was working and ready to merge into the main branch without issue.

[S6] Unit testing was carried out by writing test functions along-side the functions of the program and running these using the Visual Studio test and debugging environment. This allowed me to test whilst writing code, ensuring that no problems would be missed.
An example of one of the unit tests is that of the ExcelToCSV function, in which I am expecting to pass a FileStream, and a string indicating where the outputted CSV file should be placed (See Fig. 8). I had set up test functions to test expected and valid data, and a test to check that an error is thrown when invalid data is passed to the function. (See fig. 9)

[S4] Whilst programming, due to the fact I carried out manual testing whilst writing each stage of the program I had not encountered any failed tests due to my code not being correct. However I did encounter an error with a test due to the fact that the output file included a new line at the end of a CSV file (see fig. 19), which I was unaware of being a feature of the module I had used. My test was originally expecting the output file to be one line without any new line characters at the end. To resolve this I had changed my test code to include a new line in the expected result. (See fig. 20)

Deploy

[S10] The deployment phase involved setting up the production environment followed by running GitHub actions to deploy the application files to the production server.

Setting up the production environment was relatively simple due to there already being a suitable production server up and running for applications like this one. To create the environment we will be publishing to, I had to log on to the server using domain admin credentials, which are stored on a 256bit encrypted smart card, I then created a new IIS site with the relevant information filled in, such as the open ports (80 HTTP & 443 HTTPS), the URL "fsimport.gwayre.co.uk" (see fig. 6) and the root folder location, which I kept as the default in "C:\inetpub\wwwroot\fsimport.gwayre.co.uk\".
Finally to finish up the setup of the production environment, since we needed to access a server IP address but resolved by using a URL I needed to configure a DNS (Domain Name Server) A record (see fig. 7) which points the domain to the application server IP address. Since this site is only internal I did not need to create any firewall or NAT (Network Address Translation) rules to allow external connections.

Deploying the program was carried out semi-automatically via GitHub CI/CD in which we had set up a publish action that used a configuration file to locally push the ASP.NET application to the previously set-up IIS site on our application server.

Maintain

As of now, there has not been any active maintenance required to be done for the project, however as the program gets used by end users, I expect there to be improvements that can be made. We internally host a ticketing server in which we can accept bug fix requests, or feature requests in which we can follow the proper procedures to plan, design and implement any of these new features of bug fixes that we feel works within the bounds of the project. An advantage of the CI/CD system that we have in place is that each feature or bug fix may be pushed to production semi-automatically via our GitHub actions.

Appendix

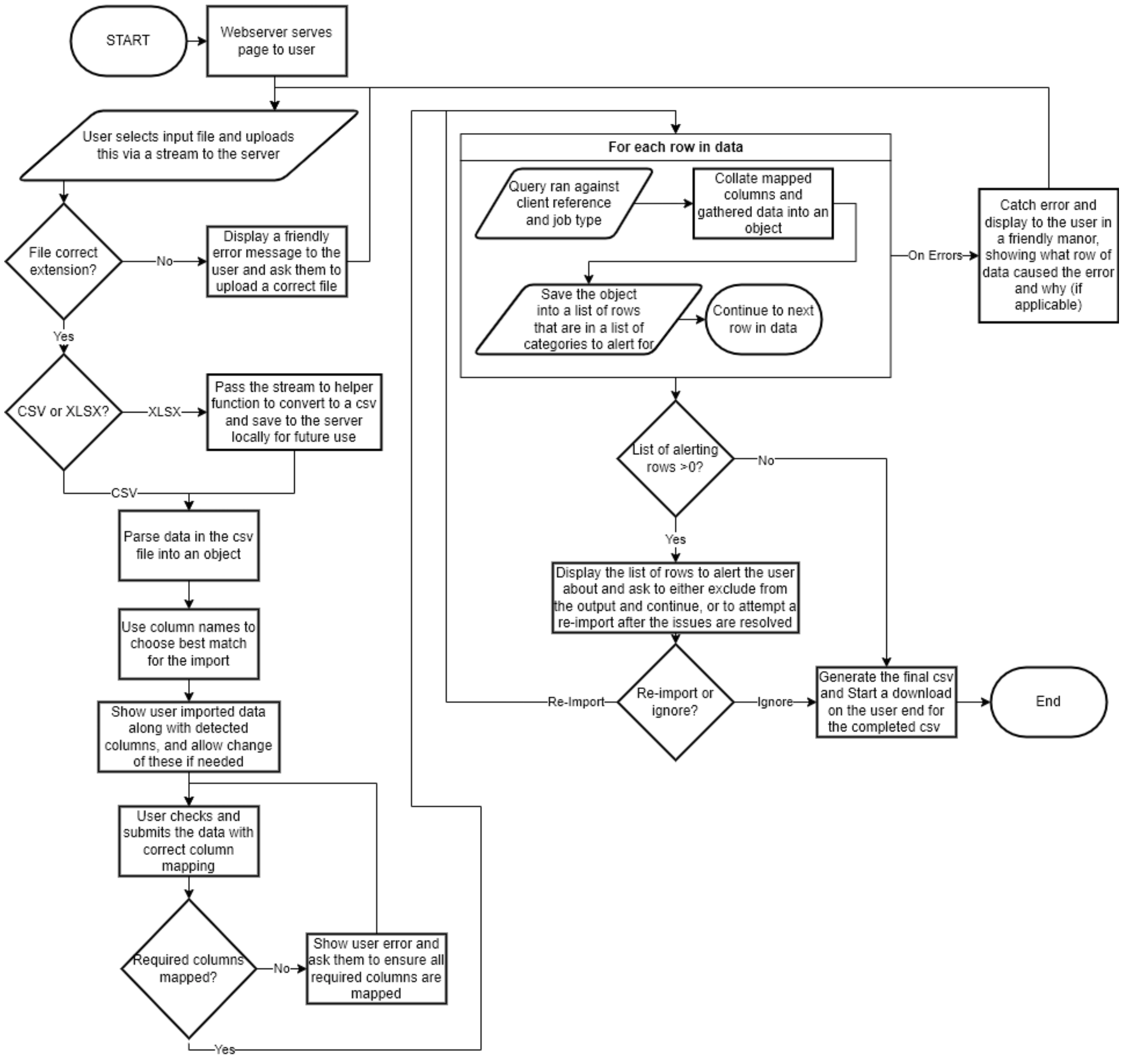
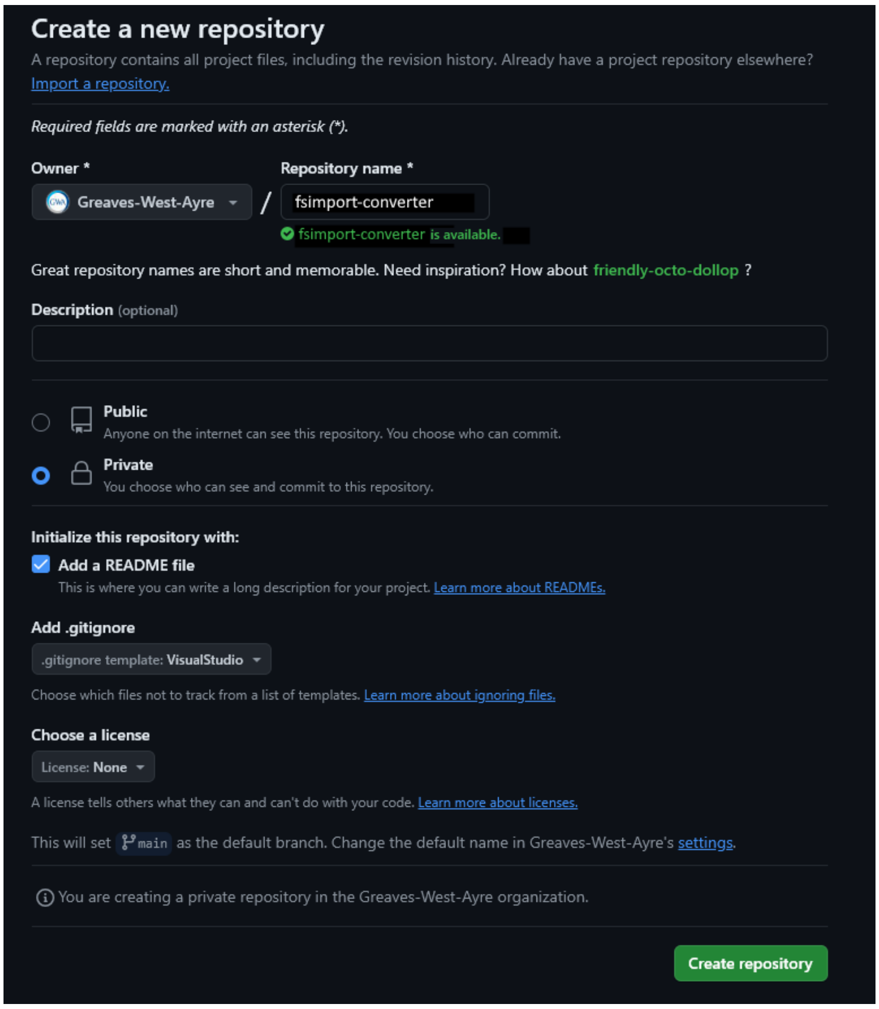

START
Webserver serves page to user
User selects input file and uploads this via a stream to the server
File correct extension?
No
Display a friendly error message to the user and ask them to upload a correct file
Yes
CSV or XLSX?
XLSX
Pass the stream to helper function to convert to a csv and save to the server locally for future use
CSV
Parse data in the csv file into an object
Use column names to choose best match for the import
Show user imported data along with detected columns, and allow change of these if needed
User checks and submits the data with correct column mapping
Required columns mapped?
No
Show user error and ask them to ensure all required columns are mapped
Yes
For each row in data
Query ran against client reference and job type
Collate mapped columns and gathered data into an object
Save the object into a list of rows that are in a list of categories to alert for
Continue to next row in data
On Errors
Catch error and display to the user in a friendly manor, showing what row of data caused the error and why (if applicable)
List of alerting rows >0?
No
Yes
Display the list of rows to alert the user about and ask to either exclude from the output and continue, or to attempt a re-import after the issues are resolved
Re-import or ignore?
Re-Import
Ignore
Generate the final csv and Start a download on the user end for the completed csv
End

Figure 1 – Flowchart diagram of the design of the program flow and how the end product should work.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner *: Greaves-West-Ayre / Repository name *: fsimport-converter [redacted]

✓ fsimport-converter is available. [redacted]

Great repository names are short and memorable. Need inspiration? How about friendly-octo-dollop ?

Description (optional):

○ Public
Anyone on the internet can see this repository. You choose who can commit.

● Private
You choose who can see and commit to this repository.

Initialize this repository with:

☑ Add a README file
This is where you can write a long description for your project. Learn more about READMEs.

Add .gitignore
.gitignore template: VisualStudio

Choose which files not to track from a list of templates. Learn more about ignoring files.

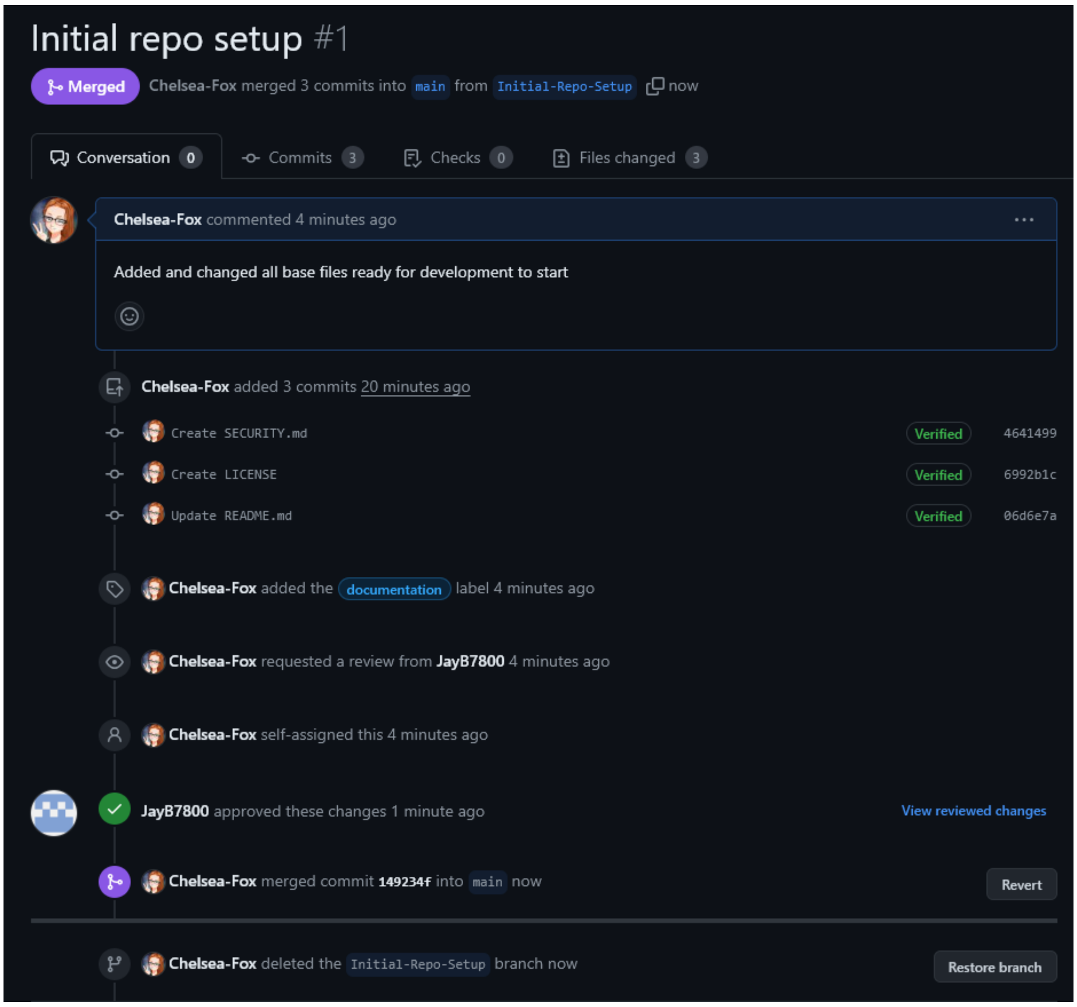
Choose a license
License: None

A license tells others what they can and can't do with your code. Learn more about licenses.

This will set `main` as the default branch. Change the default name in Greaves-West-Ayre's settings.

You are creating a private repository in the Greaves-West-Ayre organization.

Figure 2 – Screenshot showing the creation of a new GitHub repository with basic settings configured.


Initial repo setup #1

Merged Chelsea-Fox merged 3 commits into `main` from `Initial-Repo-Setup` now

Conversation 0 | Commits 3 | Checks 0 | Files changed 3

Chelsea-Fox commented 4 minutes ago

Added and changed all base files ready for development to start

Chelsea-Fox added 3 commits 20 minutes ago

```
Create SECURITY.md      Verified  4641499
Create LICENSE          Verified  6992b1c
Update README.md        Verified  06d6e7a
```

Chelsea-Fox added the documentation label 4 minutes ago

Chelsea-Fox requested a review from JayB7800 4 minutes ago

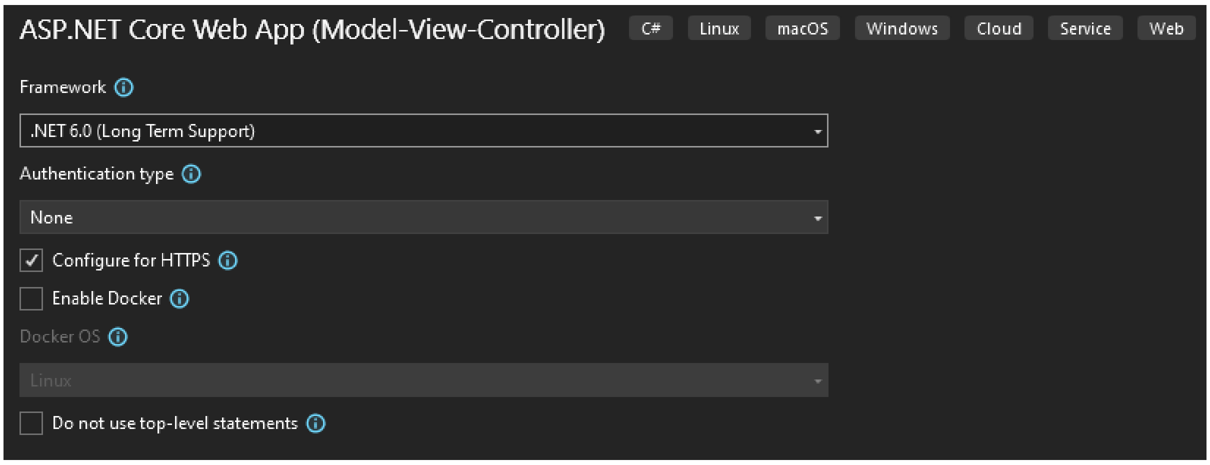
Chelsea-Fox self-assigned this 4 minutes ago

JayB7800 approved these changes 1 minute ago — View reviewed changes

Chelsea-Fox merged commit `149234f` into `main` now — Revert

Chelsea-Fox deleted the `Initial-Repo-Setup` branch now — Restore branch

Figure 3 – Screenshot of first commit to the GitHub repository, including the readme, security and licence files. Also shows the approval element required to merge commits into the main branch.


ASP.NET Core Web App (Model-View-Controller)

Framework: .NET 6.0 (Long Term Support)

Authentication type: None

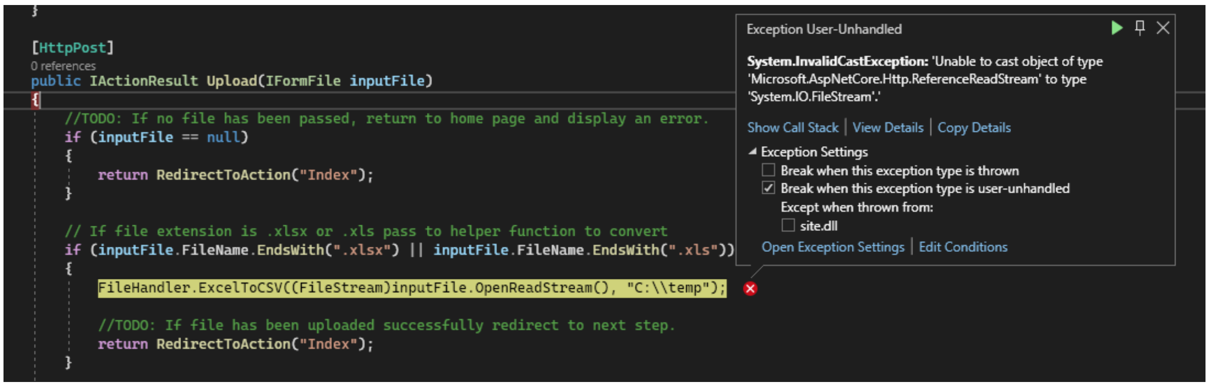
[✓] Configure for HTTPS

[ ] Enable Docker

Docker OS: Linux

[ ] Do not use top-level statements

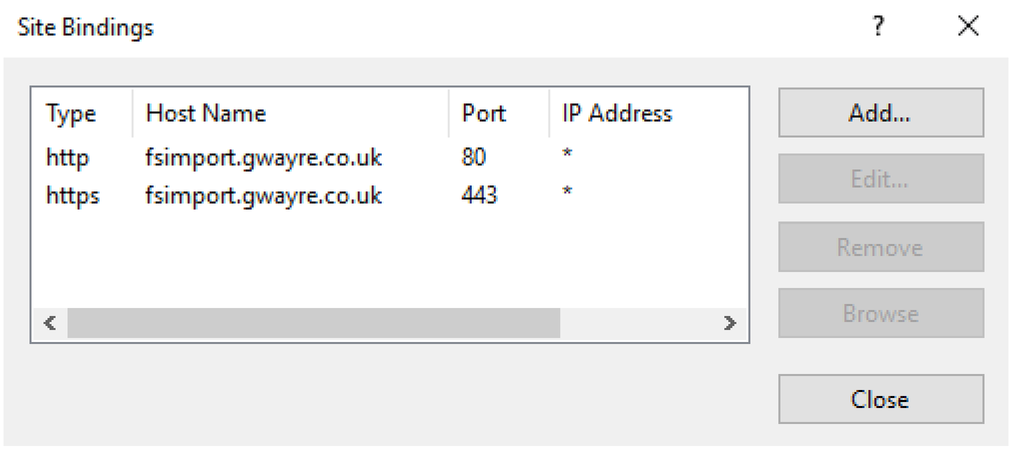
Figure 4 – Screenshot of Visual Studio. Setting up new project using the ASP.NET Core Web App (MVC) Project template.



```
}

[HttpPost]
0 references
public IActionResult Upload(IFormFile inputFile)
{
    //TODO: If no file has been passed, return to home page and display an error.
    if (inputFile == null)
    {
        return RedirectToAction("Index");
    }

    // If file extension is .xlsx or .xls pass to helper function to convert
    if (inputFile.FileName.EndsWith(".xlsx") || inputFile.FileName.EndsWith(".xls"))
    {
        FileHandler.ExcelToCSV((FileStream)inputFile.OpenReadStream(), "C:\\temp");

        //TODO: If file has been uploaded successfully redirect to next step.
        return RedirectToAction("Index");
    }
```

```
Exception User-Unhandled

System.InvalidCastException: 'Unable to cast object of type
'Microsoft.AspNetCore.Http.ReferenceReadStream' to type
'System.IO.FileStream'.'

Show Call Stack | View Details | Copy Details
Exception Settings
  [ ] Break when this exception type is thrown
  [✓] Break when this exception type is user-unhandled
      Except when thrown from:
      [ ] site.dll
Open Exception Settings | Edit Conditions
```

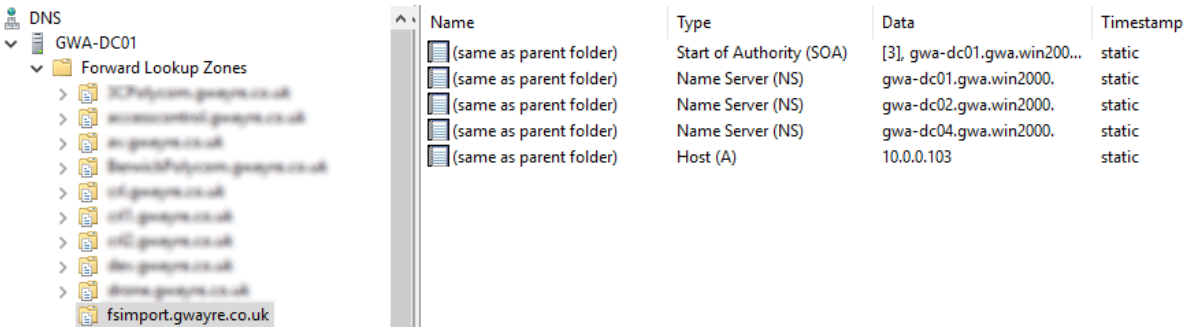
Figure 5 – Screenshot of error whilst programming and testing.



Site Bindings

Type	Host Name	Port	IP Address
http	fsimport.gwayre.co.uk	80	*
https	fsimport.gwayre.co.uk	443	*

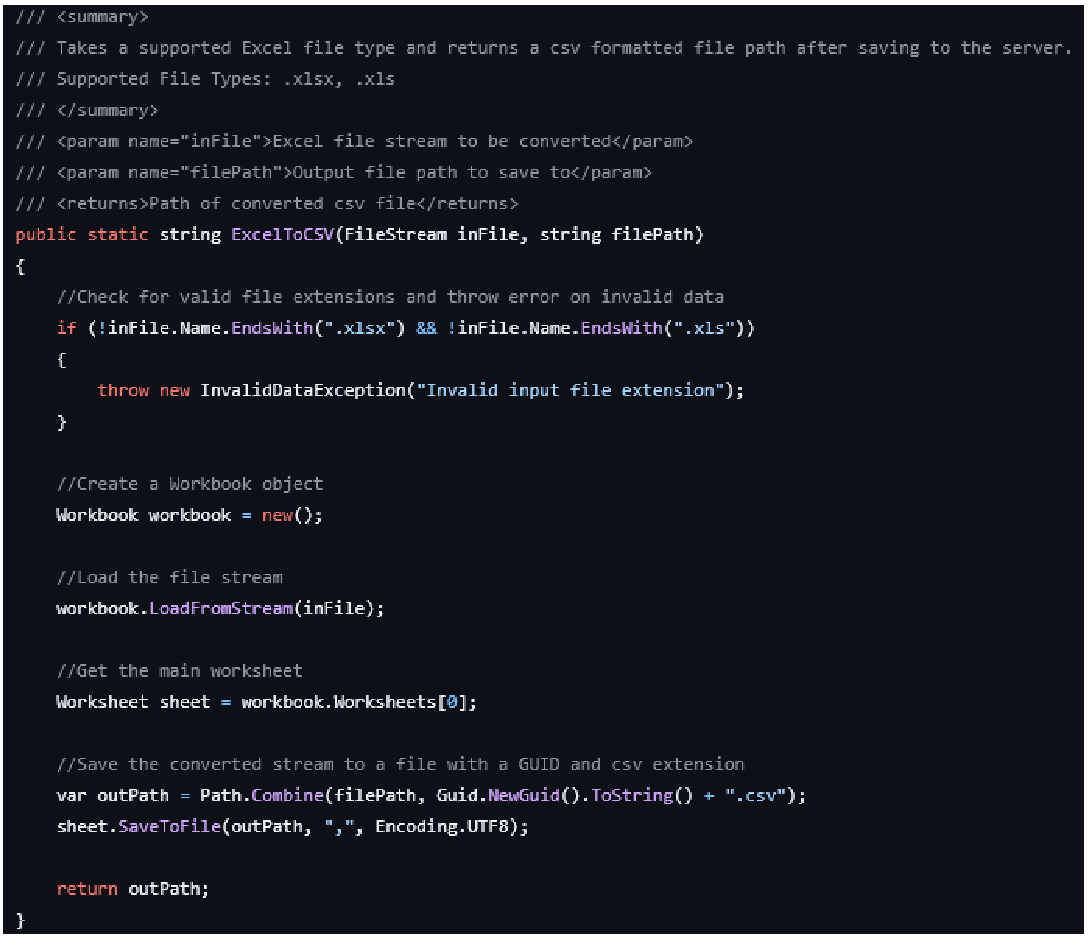
Figure 6 – Screenshot of IIS server bindings for the web app.



DNS
GWA-DC01
Forward Lookup Zones
[redacted]
fsimport.gwayre.co.uk

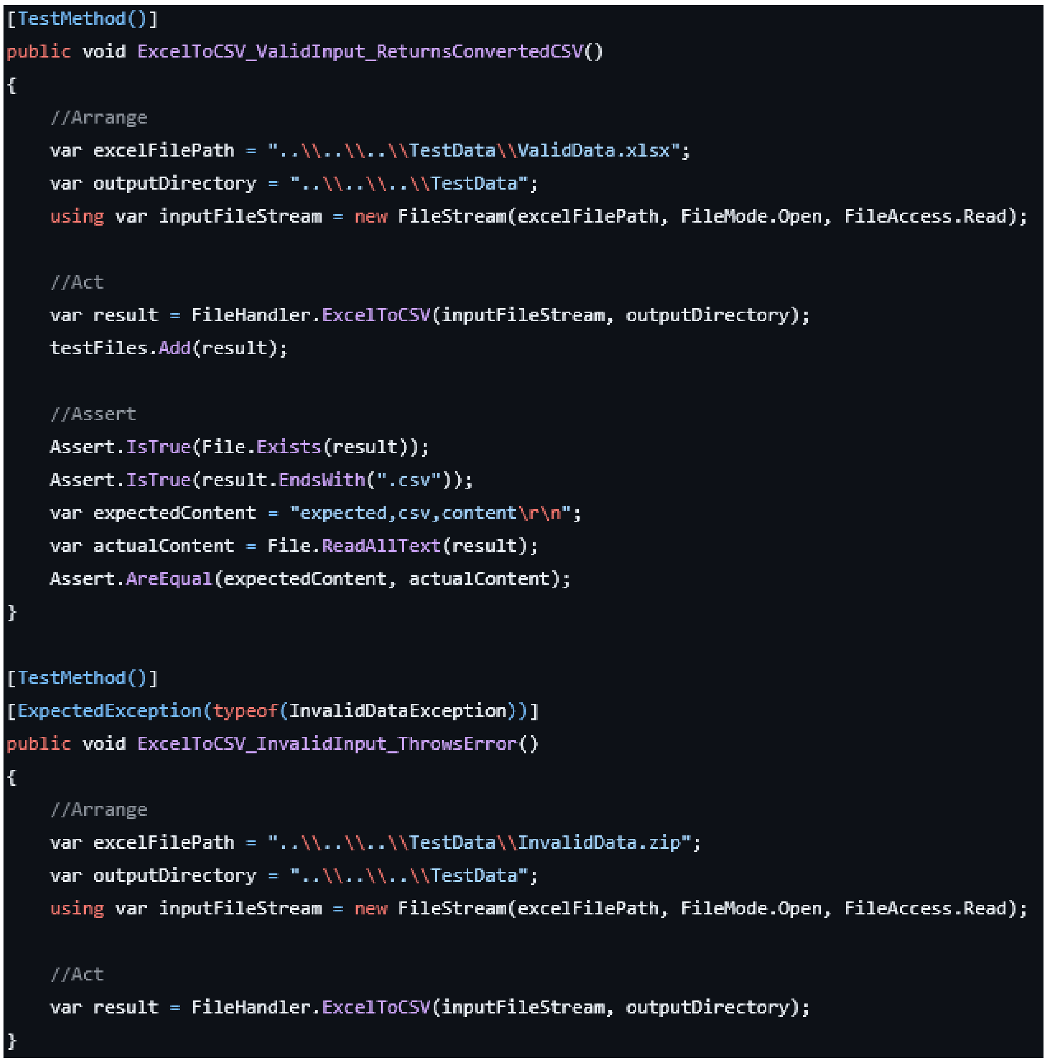
Name	Type	Data	Timestamp
(same as parent folder)	Start of Authority (SOA)	[3], gwa-dc01.gwa.win200...	static
(same as parent folder)	Name Server (NS)	gwa-dc01.gwa.win2000.	static
(same as parent folder)	Name Server (NS)	gwa-dc02.gwa.win2000.	static
(same as parent folder)	Name Server (NS)	gwa-dc04.gwa.win2000.	static
(same as parent folder)	Host (A)	10.0.0.103	static

Figure 7 – Screenshot of DNS server A record for fsimport.gwayre.co.uk.



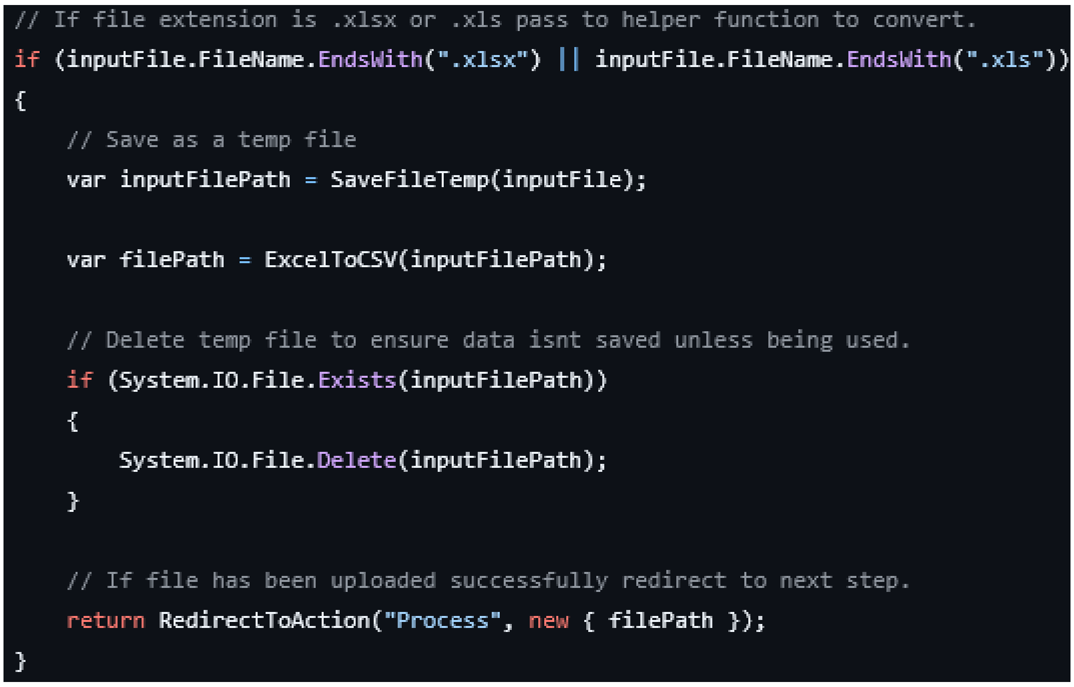
```csharp
/// <summary>
/// Takes a supported Excel file type and returns a csv formatted file path after saving to the server.
/// Supported File Types: .xlsx, .xls
/// </summary>
/// <param name="inFile">Excel file stream to be converted</param>
/// <param name="filePath">Output file path to save to</param>
/// <returns>Path of converted csv file</returns>
public static string ExcelToCSV(FileStream inFile, string filePath)
{
    //Check for valid file extensions and throw error on invalid data
    if (!inFile.Name.EndsWith(".xlsx") && !inFile.Name.EndsWith(".xls"))
    {
        throw new InvalidDataException("Invalid input file extension");
    }

    //Create a Workbook object
    Workbook workbook = new();

    //Load the file stream
    workbook.LoadFromStream(inFile);

    //Get the main worksheet
    Worksheet sheet = workbook.Worksheets[0];

    //Save the converted stream to a file with a GUID and csv extension
    var outPath = Path.Combine(filePath, Guid.NewGuid().ToString() + ".csv");
    sheet.SaveToFile(outPath, ",", Encoding.UTF8);

    return outPath;
}
```

Figure 8 – Screenshot of ExcelToCSV function, intended to convert a given Excel document to a CSV file.



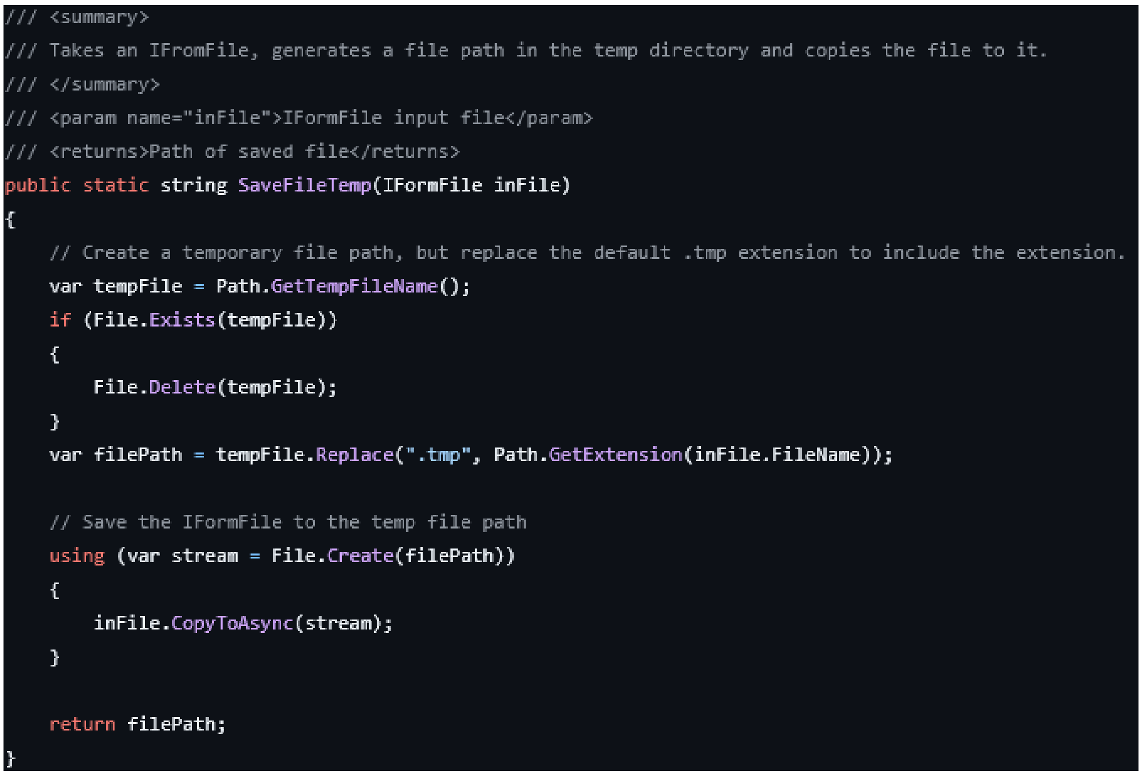
```
[TestMethod()]
public void ExcelToCSV_ValidInput_ReturnsConvertedCSV()
{
    //Arrange
    var excelFilePath = "..\\..\\..\\TestData\\ValidData.xlsx";
    var outputDirectory = "..\\..\\..\\TestData";
    using var inputFileStream = new FileStream(excelFilePath, FileMode.Open, FileAccess.Read);

    //Act
    var result = FileHandler.ExcelToCSV(inputFileStream, outputDirectory);
    testFiles.Add(result);

    //Assert
    Assert.IsTrue(File.Exists(result));
    Assert.IsTrue(result.EndsWith(".csv"));
    var expectedContent = "expected,csv,content\r\n";
    var actualContent = File.ReadAllText(result);
    Assert.AreEqual(expectedContent, actualContent);
}

[TestMethod()]
[ExpectedException(typeof(InvalidDataException))]
public void ExcelToCSV_InvalidInput_ThrowsError()
{
    //Arrange
    var excelFilePath = "..\\..\\..\\TestData\\InvalidData.zip";
    var outputDirectory = "..\\..\\..\\TestData";
    using var inputFileStream = new FileStream(excelFilePath, FileMode.Open, FileAccess.Read);

    //Act
    var result = FileHandler.ExcelToCSV(inputFileStream, outputDirectory);
}
```

Figure 9 – Screenshot of unit tests written for the ExcelToCSV function.



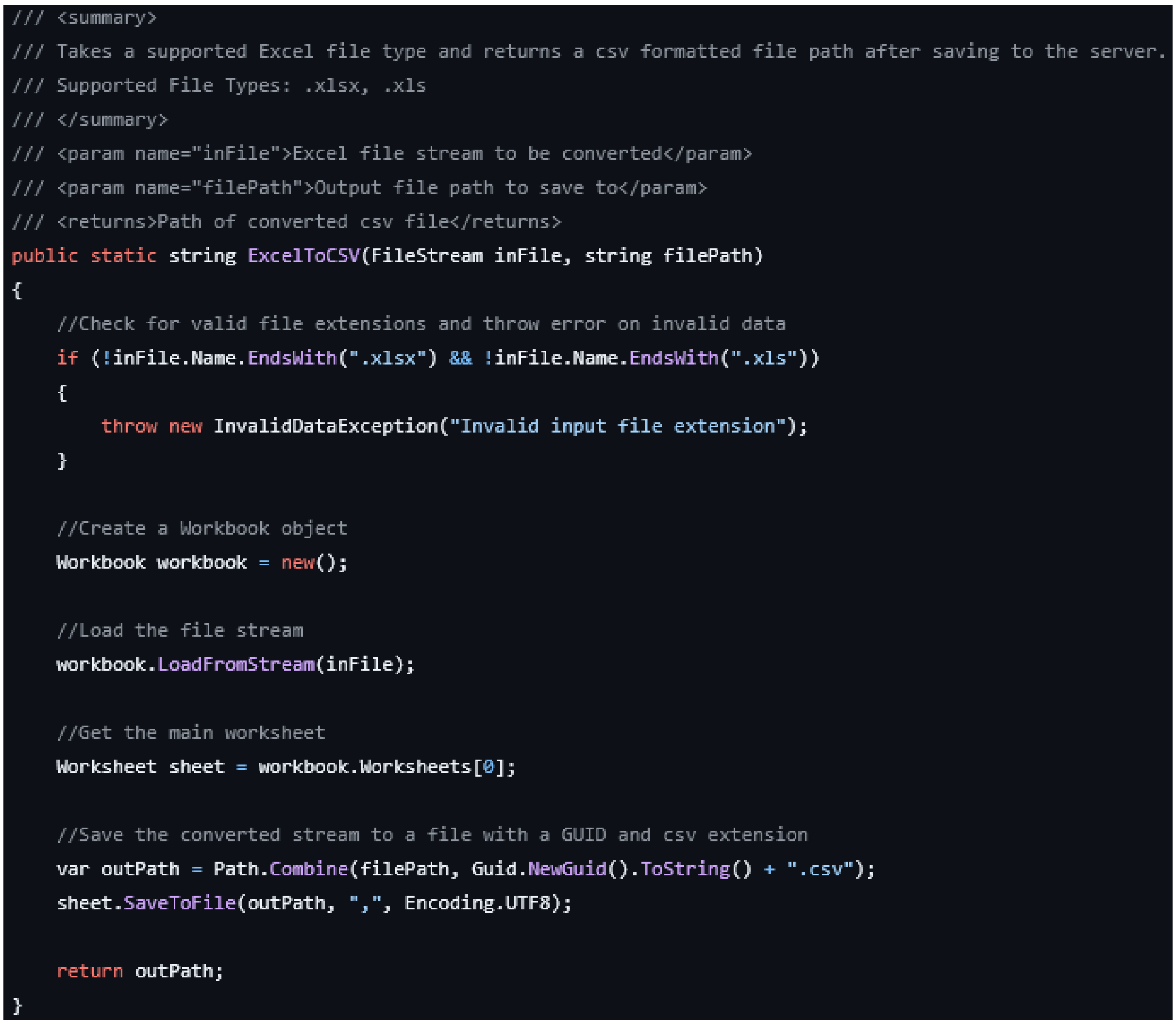
```
// If file extension is .xlsx or .xls pass to helper function to convert.
if (inputFile.FileName.EndsWith(".xlsx") || inputFile.FileName.EndsWith(".xls"))
{
    // Save as a temp file
    var inputFilePath = SaveFileTemp(inputFile);

    var filePath = ExcelToCSV(inputFilePath);

    // Delete temp file to ensure data isnt saved unless being used.
    if (System.IO.File.Exists(inputFilePath))
    {
        System.IO.File.Delete(inputFilePath);
    }

    // If file has been uploaded successfully redirect to next step.
    return RedirectToAction("Process", new { filePath });
}
```

Figure 10 – Screenshot showing code that saves the input file as a temporary file before then passing this to the ExcelToCSV function.



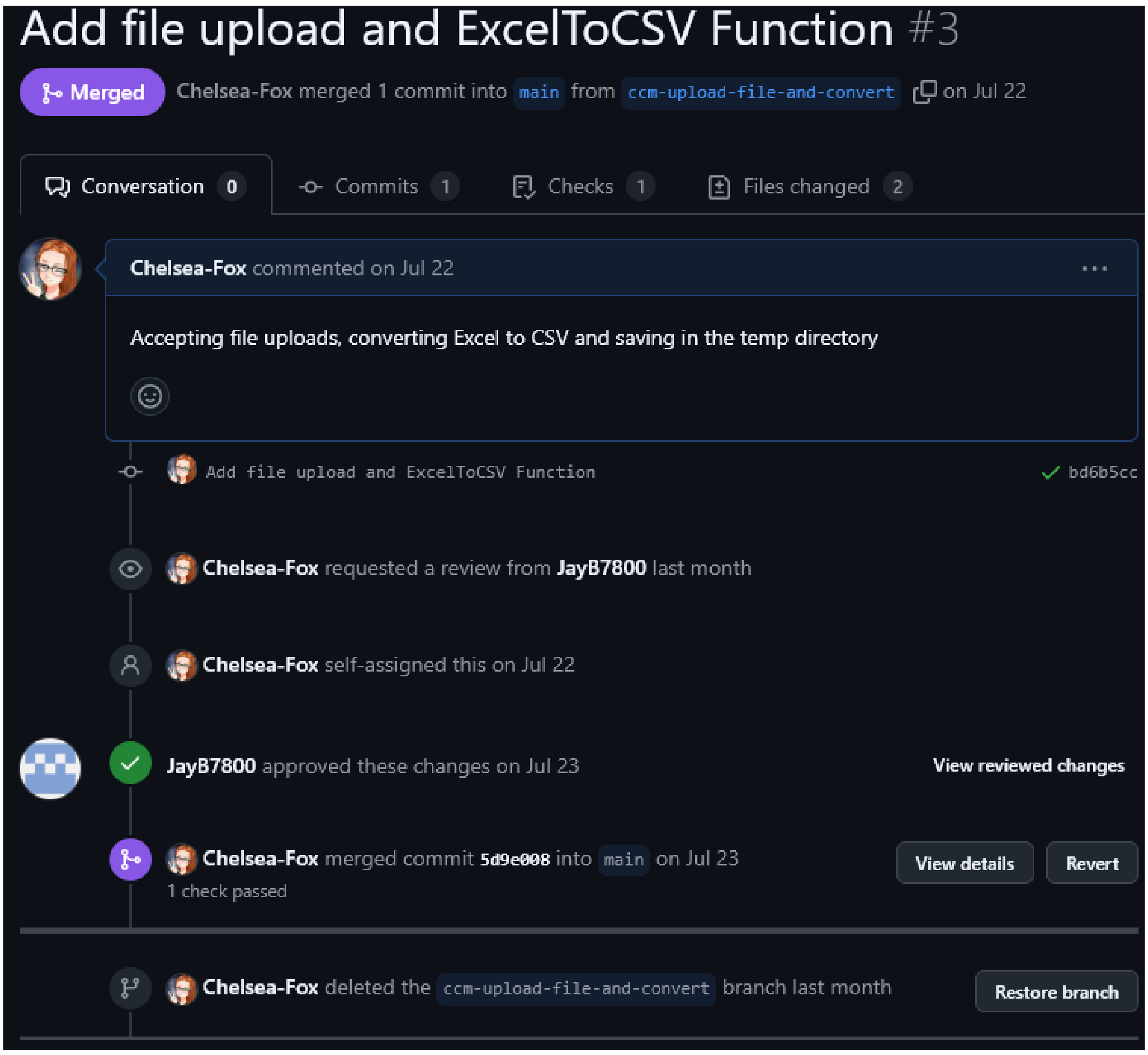
```
/// <summary>
/// Takes an IFromFile, generates a file path in the temp directory and copies the file to it.
/// </summary>
/// <param name="inFile">IFormFile input file</param>
/// <returns>Path of saved file</returns>
public static string SaveFileTemp(IFormFile inFile)
{
    // Create a temporary file path, but replace the default .tmp extension to include the extension.
    var tempFile = Path.GetTempFileName();
    if (File.Exists(tempFile))
    {
        File.Delete(tempFile);
    }
    var filePath = tempFile.Replace(".tmp", Path.GetExtension(inFile.FileName));

    // Save the IFormFile to the temp file path
    using (var stream = File.Create(filePath))
    {
        inFile.CopyToAsync(stream);
    }

    return filePath;
}
```

Figure 11 – Temporary file saving function by using the inbuilt GetTempFileName from the Path library.



```
/// <summary>
/// Takes a supported Excel file type and returns a csv formatted file path after saving to the server.
/// Supported File Types: .xlsx, .xls
/// </summary>
/// <param name="inFile">Excel file stream to be converted</param>
/// <param name="filePath">Output file path to save to</param>
/// <returns>Path of converted csv file</returns>
public static string ExcelToCSV(FileStream inFile, string filePath)
{
    //Check for valid file extensions and throw error on invalid data
    if (!inFile.Name.EndsWith(".xlsx") && !inFile.Name.EndsWith(".xls"))
    {
        throw new InvalidDataException("Invalid input file extension");
    }

    //Create a Workbook object
    Workbook workbook = new();

    //Load the file stream
    workbook.LoadFromStream(inFile);

    //Get the main worksheet
    Worksheet sheet = workbook.Worksheets[0];

    //Save the converted stream to a file with a GUID and csv extension
    var outPath = Path.Combine(filePath, Guid.NewGuid().ToString() + ".csv");
    sheet.SaveToFile(outPath, ",", Encoding.UTF8);

    return outPath;
}
```

Figure 12 – ExcelToCSV function showing use of the, NuGet package, Spire.XLS workbook functionality.


Add file upload and ExcelToCSV Function #3

Merged Chelsea-Fox merged 1 commit into `main` from `ccm-upload-file-and-convert` on Jul 22

Chelsea-Fox commented on Jul 22

Accepting file uploads, converting Excel to CSV and saving in the temp directory

`Add file upload and ExcelToCSV Function` ✓ `bd6b5cc`

Chelsea-Fox requested a review from **JayB7800** last month

Chelsea-Fox self-assigned this on Jul 22

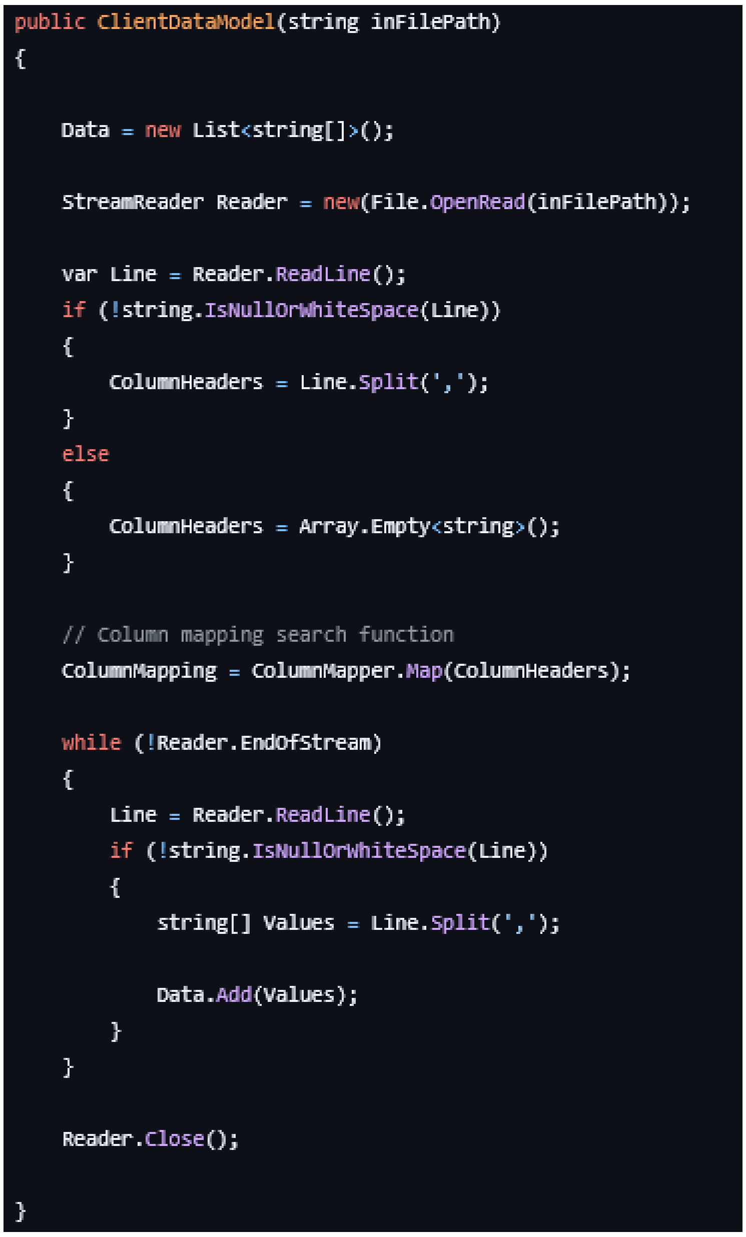
JayB7800 approved these changes on Jul 23

Chelsea-Fox merged commit `5d9e008` into `main` on Jul 23

1 check passed

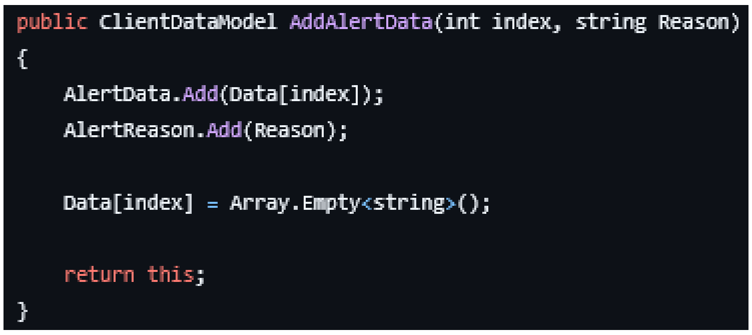
Chelsea-Fox deleted the `ccm-upload-file-and-convert` branch last month

Figure 13 – Screenshot showing a GitHub commit along with approval of the commit by a colleague.



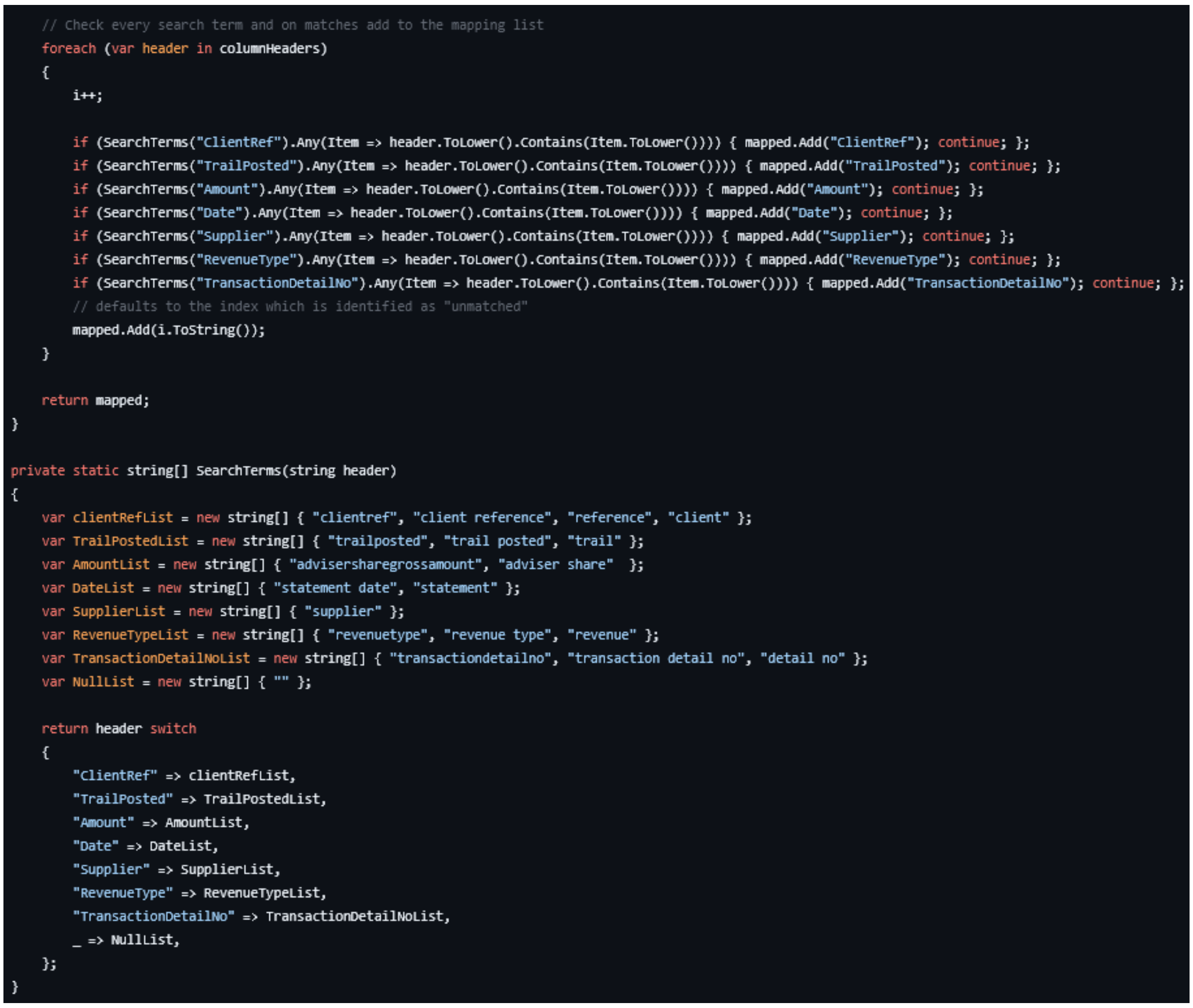
```
public ClientDataModel(string inFilePath)
{

    Data = new List<string[]>();

    StreamReader Reader = new(File.OpenRead(inFilePath));

    var Line = Reader.ReadLine();
    if (!string.IsNullOrWhiteSpace(Line))
    {
        ColumnHeaders = Line.Split(',');
    }
    else
    {
        ColumnHeaders = Array.Empty<string>();
    }

    // Column mapping search function
    ColumnMapping = ColumnMapper.Map(ColumnHeaders);

    while (!Reader.EndOfStream)
    {
        Line = Reader.ReadLine();
        if (!string.IsNullOrWhiteSpace(Line))
        {
            string[] Values = Line.Split(',');

            Data.Add(Values);
        }
    }

    Reader.Close();

}
```

Figure 14 – Screenshot of the ClientDataModel class and its constructor.



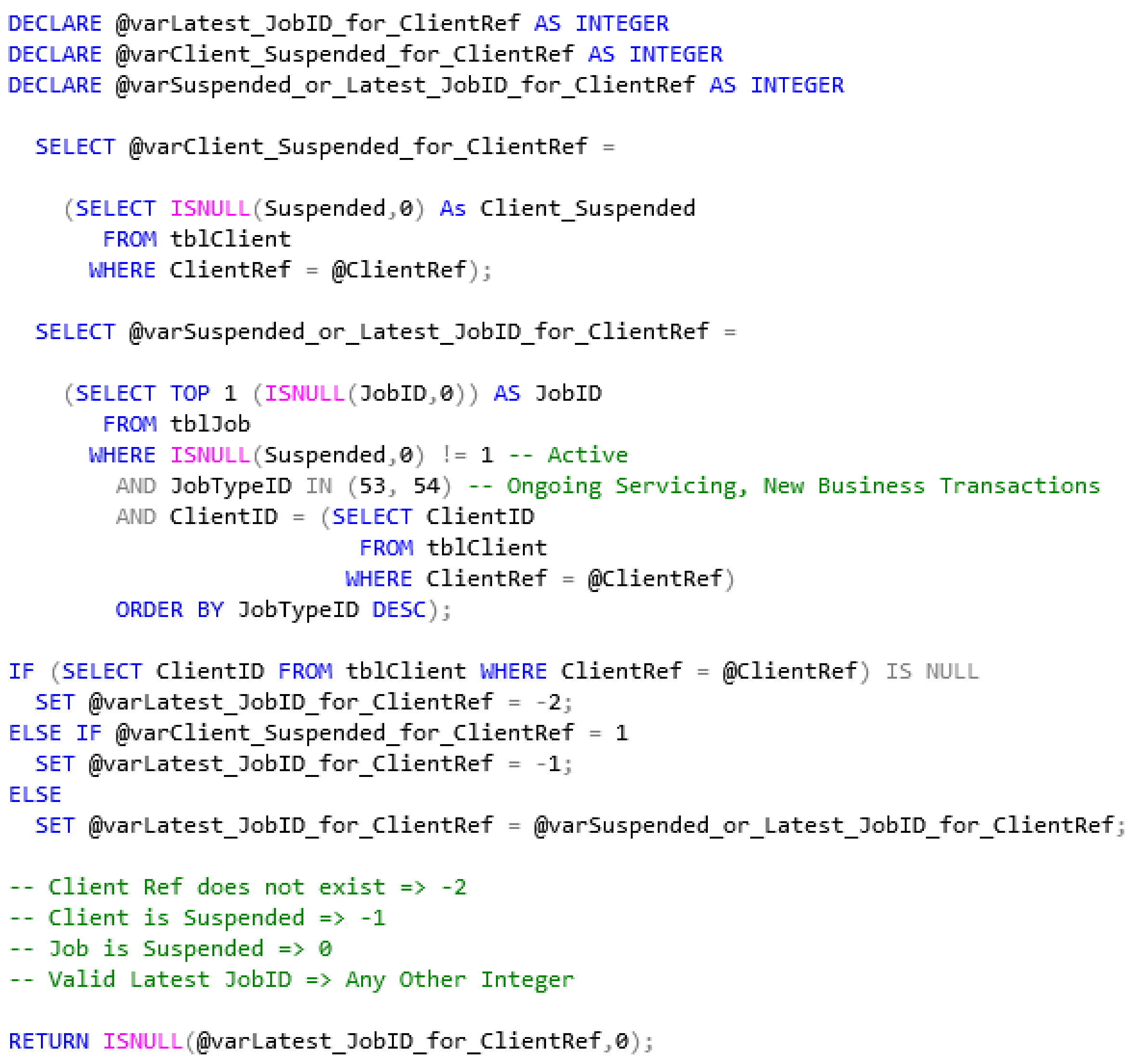
```
public ClientDataModel AddAlertData(int index, string Reason)
{
    AlertData.Add(Data[index]);
    AlertReason.Add(Reason);

    Data[index] = Array.Empty<string>();

    return this;
}
```

Figure 15 – Screenshot of a class method to add data to the alert list for later use by the program.



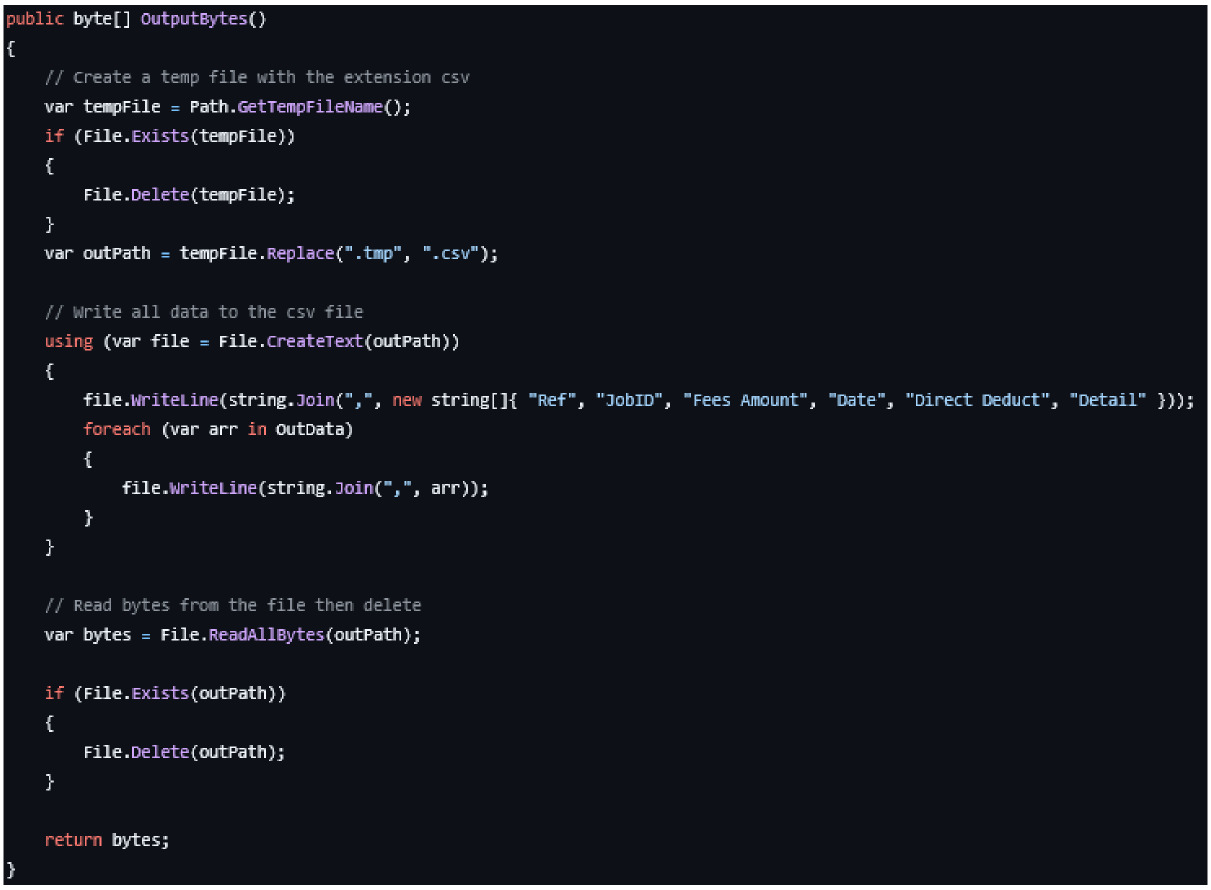
```
    // Check every search term and on matches add to the mapping list
    foreach (var header in columnHeaders)
    {
        i++;

        if (SearchTerms("ClientRef").Any(Item => header.ToLower().Contains(Item.ToLower()))) { mapped.Add("ClientRef"); continue; };
        if (SearchTerms("TrailPosted").Any(Item => header.ToLower().Contains(Item.ToLower()))) { mapped.Add("TrailPosted"); continue; };
        if (SearchTerms("Amount").Any(Item => header.ToLower().Contains(Item.ToLower()))) { mapped.Add("Amount"); continue; };
        if (SearchTerms("Date").Any(Item => header.ToLower().Contains(Item.ToLower()))) { mapped.Add("Date"); continue; };
        if (SearchTerms("Supplier").Any(Item => header.ToLower().Contains(Item.ToLower()))) { mapped.Add("Supplier"); continue; };
        if (SearchTerms("RevenueType").Any(Item => header.ToLower().Contains(Item.ToLower()))) { mapped.Add("RevenueType"); continue; };
        if (SearchTerms("TransactionDetailNo").Any(Item => header.ToLower().Contains(Item.ToLower()))) { mapped.Add("TransactionDetailNo"); continue; };
        // defaults to the index which is identified as "unmatched"
        mapped.Add(i.ToString());
    }

    return mapped;
}

private static string[] SearchTerms(string header)
{
    var clientRefList = new string[] { "clientref", "client reference", "reference", "client" };
    var TrailPostedList = new string[] { "trailposted", "trail posted", "trail" };
    var AmountList = new string[] { "advisersharegrossamount", "adviser share"  };
    var DateList = new string[] { "statement date", "statement" };
    var SupplierList = new string[] { "supplier" };
    var RevenueTypeList = new string[] { "revenuetype", "revenue type", "revenue" };
    var TransactionDetailNoList = new string[] { "transactiondetailno", "transaction detail no", "detail no" };
    var NullList = new string[] { "" };

    return header switch
    {
        "ClientRef" => clientRefList,
        "TrailPosted" => TrailPostedList,
        "Amount" => AmountList,
        "Date" => DateList,
        "Supplier" => SupplierList,
        "RevenueType" => RevenueTypeList,
        "TransactionDetailNo" => TransactionDetailNoList,
        _ => NullList,
    };
}
```

Figure 16 – Screenshot of the column mapper function alongside the statically coded search terms.



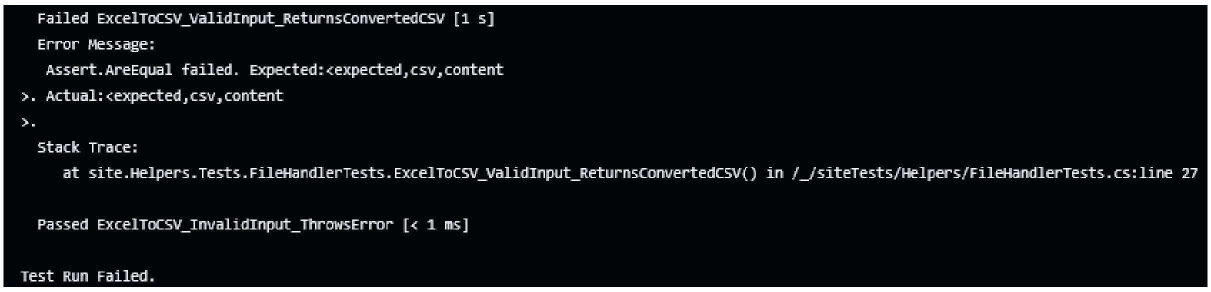
```sql
DECLARE @varLatest_JobID_for_ClientRef AS INTEGER
DECLARE @varClient_Suspended_for_ClientRef AS INTEGER
DECLARE @varSuspended_or_Latest_JobID_for_ClientRef AS INTEGER

  SELECT @varClient_Suspended_for_ClientRef =

    (SELECT ISNULL(Suspended,0) As Client_Suspended
      FROM tblClient
     WHERE ClientRef = @ClientRef);

  SELECT @varSuspended_or_Latest_JobID_for_ClientRef =

    (SELECT TOP 1 (ISNULL(JobID,0)) AS JobID
      FROM tblJob
     WHERE ISNULL(Suspended,0) != 1 -- Active
       AND JobTypeID IN (53, 54) -- Ongoing Servicing, New Business Transactions
       AND ClientID = (SELECT ClientID
                        FROM tblClient
                       WHERE ClientRef = @ClientRef)
       ORDER BY JobTypeID DESC);

IF (SELECT ClientID FROM tblClient WHERE ClientRef = @ClientRef) IS NULL
  SET @varLatest_JobID_for_ClientRef = -2;
ELSE IF @varClient_Suspended_for_ClientRef = 1
  SET @varLatest_JobID_for_ClientRef = -1;
ELSE
  SET @varLatest_JobID_for_ClientRef = @varSuspended_or_Latest_JobID_for_ClientRef;

-- Client Ref does not exist => -2
-- Client is Suspended => -1
-- Job is Suspended => 0
-- Valid Latest JobID => Any Other Integer

RETURN ISNULL(@varLatest_JobID_for_ClientRef,0);
```

Figure 17 – Screenshot of the SQL function that was used in the programs data access layer.



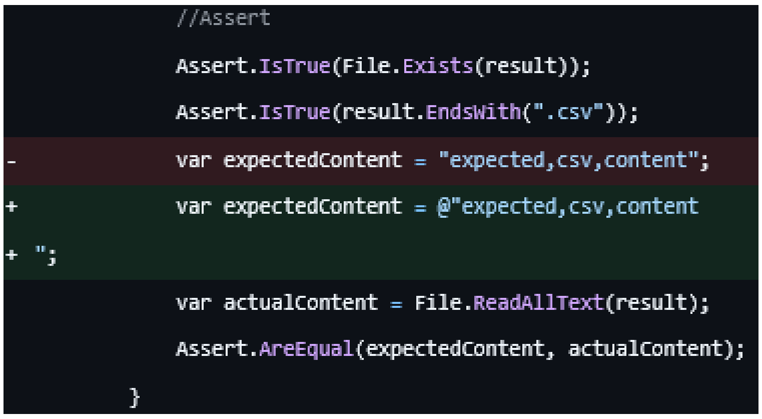
```csharp
public byte[] OutputBytes()
{
    // Create a temp file with the extension csv
    var tempFile = Path.GetTempFileName();
    if (File.Exists(tempFile))
    {
        File.Delete(tempFile);
    }
    var outPath = tempFile.Replace(".tmp", ".csv");

    // Write all data to the csv file
    using (var file = File.CreateText(outPath))
    {
        file.WriteLine(string.Join(",", new string[]{ "Ref", "JobID", "Fees Amount", "Date", "Direct Deduct", "Detail" }));
        foreach (var arr in OutData)
        {
            file.WriteLine(string.Join(",", arr));
        }
    }

    // Read bytes from the file then delete
    var bytes = File.ReadAllBytes(outPath);

    if (File.Exists(outPath))
    {
        File.Delete(outPath);
    }

    return bytes;
}
```

Figure 18 – Screenshot of the class method that outputs the bytes of the formatted CSV file.



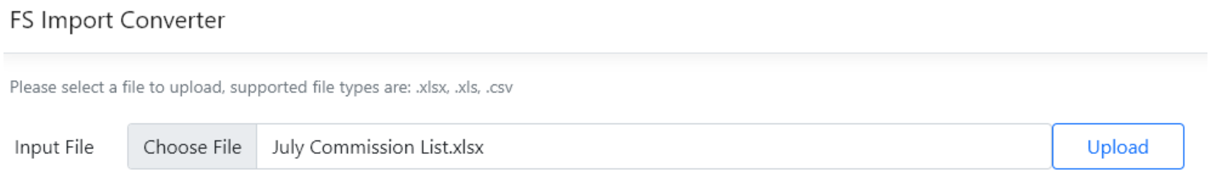
```
   Failed ExcelToCSV_ValidInput_ReturnsConvertedCSV [1 s]
   Error Message:
    Assert.AreEqual failed. Expected:<expected,csv,content
 >. Actual:<expected,csv,content
 >.
   Stack Trace:
      at site.Helpers.Tests.FileHandlerTests.ExcelToCSV_ValidInput_ReturnsConvertedCSV() in /_/siteTests/Helpers/FileHandlerTests.cs:line 27

   Passed ExcelToCSV_InvalidInput_ThrowsError [< 1 ms]

 Test Run Failed.
```

Figure 19 – Screenshot of CI/CD test results showing that there was an error with the ExcelToCSV function.



```
            //Assert
            Assert.IsTrue(File.Exists(result));
            Assert.IsTrue(result.EndsWith(".csv"));
-           var expectedContent = "expected,csv,content";
+           var expectedContent = @"expected,csv,content
+ ";
            var actualContent = File.ReadAllText(result);
            Assert.AreEqual(expectedContent, actualContent);
        }
```

Figure 20 – Screenshot of changes made to the assertion statement for the ExcelToCSV Test, including a newline.

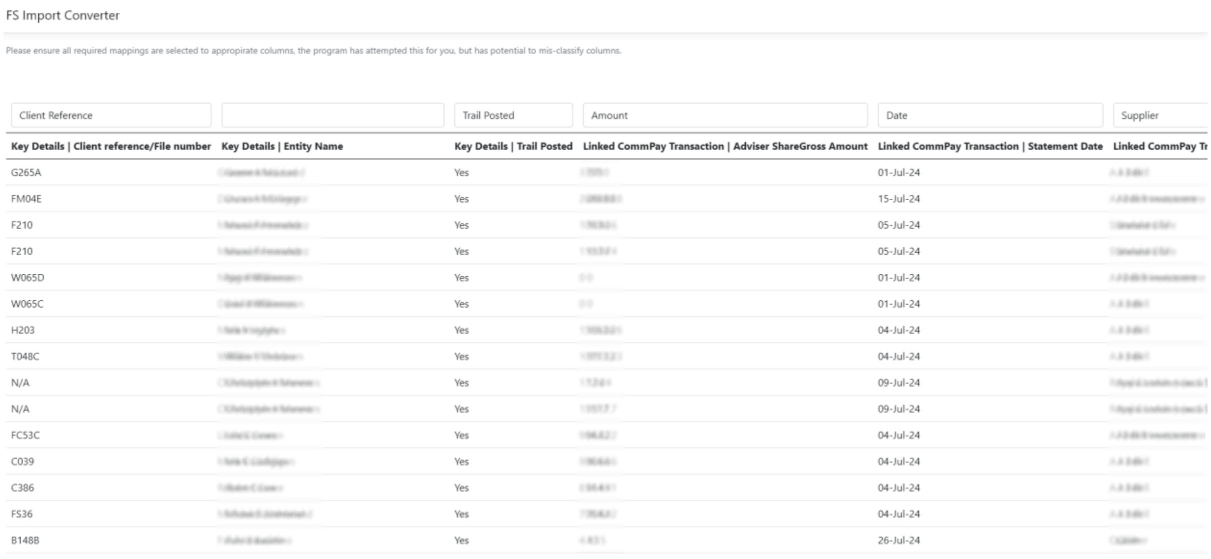


FS Import Converter

Please select a file to upload, supported file types are: .xlsx, .xls, .csv

Input File | Choose File | July Commission List.xlsx | Upload

Figure 21 – End User UI of the program asking for a file to be uploaded.

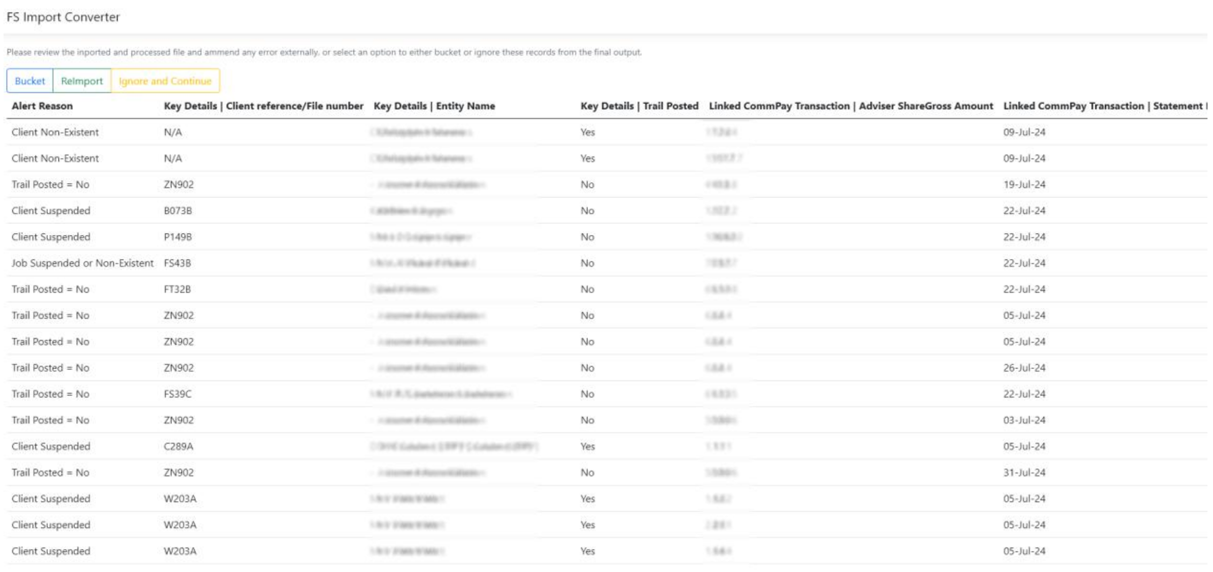


FS Import Converter

Please ensure all required mappings are selected to appropriate columns, the program has attempted this for you, but has potential to mis-classify columns.

Client Reference		Trail Posted	Amount	Date	Supplier
Key Details | Client reference/File number	**Key Details | Entity Name**	**Key Details | Trail Posted**	**Linked CommPay Transaction | Adviser ShareGross Amount**	**Linked CommPay Transaction | Statement Date**	**Linked CommPay Tr**
G265A	[redacted]	Yes	[redacted]	01-Jul-24	[redacted]
FM04E	[redacted]	Yes	[redacted]	15-Jul-24	[redacted]
F210	[redacted]	Yes	[redacted]	05-Jul-24	[redacted]
F210	[redacted]	Yes	[redacted]	05-Jul-24	[redacted]
W065D	[redacted]	Yes	[redacted]	01-Jul-24	[redacted]
W065C	[redacted]	Yes	[redacted]	01-Jul-24	[redacted]
H203	[redacted]	Yes	[redacted]	04-Jul-24	[redacted]
T048C	[redacted]	Yes	[redacted]	04-Jul-24	[redacted]
N/A	[redacted]	Yes	[redacted]	09-Jul-24	[redacted]
N/A	[redacted]	Yes	[redacted]	09-Jul-24	[redacted]
FC53C	[redacted]	Yes	[redacted]	04-Jul-24	[redacted]
C039	[redacted]	Yes	[redacted]	04-Jul-24	[redacted]
C386	[redacted]	Yes	[redacted]	04-Jul-24	[redacted]
FS36	[redacted]	Yes	[redacted]	04-Jul-24	[redacted]
B148B	[redacted]	Yes	[redacted]	26-Jul-24	[redacted]

Figure 22 – End User UI of the program having automatically mapped columns and showing the input file for user checking.

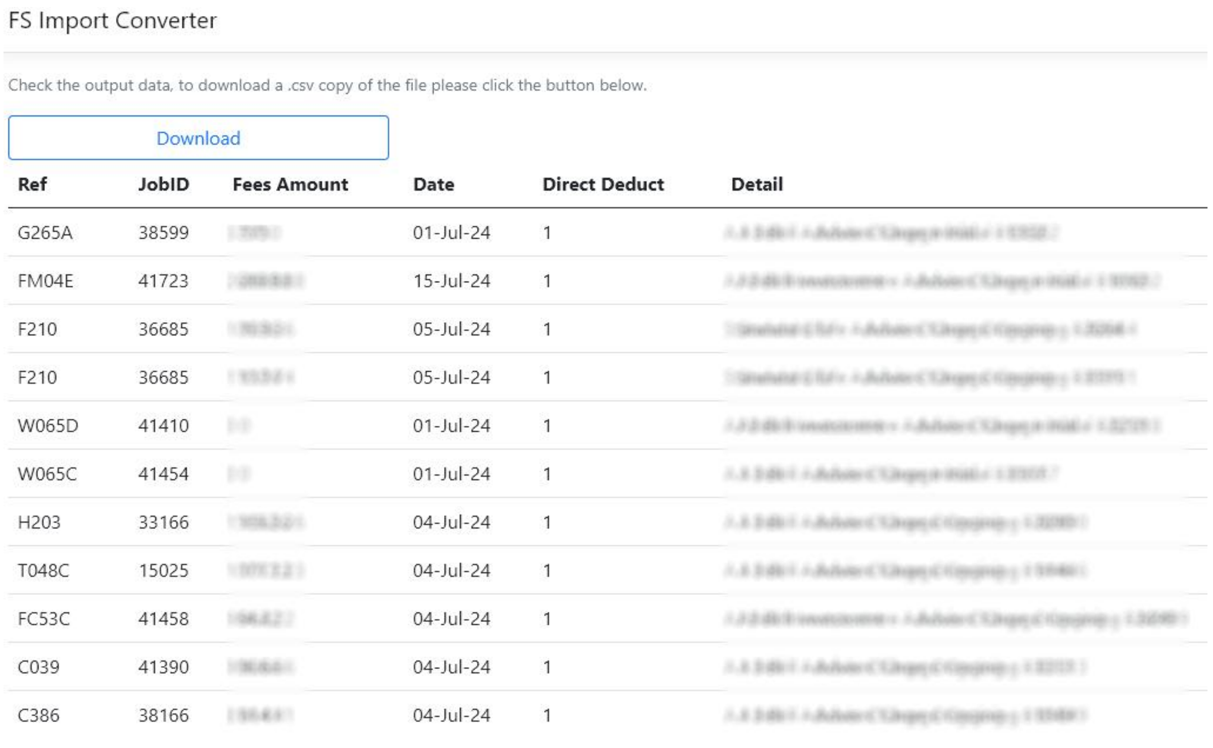


FS Import Converter

Please review the inported and processed file and ammend any error externally, or select an option to either bucket or ignore these records from the final output.

Alert Reason	Key Details | Client reference/File number	Key Details | Entity Name	Key Details | Trail Posted	Linked CommPay Transaction | Adviser ShareGross Amount	Linked CommPay Transaction | Statement
Client Non-Existent	N/A	[illegible]	Yes	[illegible]	09-Jul-24
Client Non-Existent	N/A	[illegible]	Yes	[illegible]	09-Jul-24
Trail Posted = No	ZN902	[illegible]	No	[illegible]	19-Jul-24
Client Suspended	B073B	[illegible]	No	[illegible]	22-Jul-24
Client Suspended	P149B	[illegible]	No	[illegible]	22-Jul-24
Job Suspended or Non-Existent	FS43B	[illegible]	No	[illegible]	22-Jul-24
Trail Posted = No	FT32B	[illegible]	No	[illegible]	22-Jul-24
Trail Posted = No	ZN902	[illegible]	No	[illegible]	05-Jul-24
Trail Posted = No	ZN902	[illegible]	No	[illegible]	05-Jul-24
Trail Posted = No	ZN902	[illegible]	No	[illegible]	26-Jul-24
Trail Posted = No	FS39C	[illegible]	No	[illegible]	22-Jul-24
Trail Posted = No	ZN902	[illegible]	No	[illegible]	03-Jul-24
Client Suspended	C289A	[illegible]	Yes	[illegible]	05-Jul-24
Trail Posted = No	ZN902	[illegible]	No	[illegible]	31-Jul-24
Client Suspended	W203A	[illegible]	Yes	[illegible]	05-Jul-24
Client Suspended	W203A	[illegible]	Yes	[illegible]	05-Jul-24
Client Suspended	W203A	[illegible]	Yes	[illegible]	05-Jul-24

Figure 23 – End User UI of the program showing errors that the program had found by running a SQL script against the rows of data, asking the user to bucket, re-import or ignore.



FS Import Converter

Check the output data, to download a .csv copy of the file please click the button below.

Ref	JobID	Fees Amount	Date	Direct Deduct	Detail
G265A	38599	[illegible]	01-Jul-24	1	[illegible]
FM04E	41723	[illegible]	15-Jul-24	1	[illegible]
F210	36685	[illegible]	05-Jul-24	1	[illegible]
F210	36685	[illegible]	05-Jul-24	1	[illegible]
W065D	41410	[illegible]	01-Jul-24	1	[illegible]
W065C	41454	[illegible]	01-Jul-24	1	[illegible]
H203	33166	[illegible]	04-Jul-24	1	[illegible]
T048C	15025	[illegible]	04-Jul-24	1	[illegible]
FC53C	41458	[illegible]	04-Jul-24	1	[illegible]
C039	41390	[illegible]	04-Jul-24	1	[illegible]
C386	38166	[illegible]	04-Jul-24	1	[illegible]

Figure 24 – End User UI of the program showing the final converted file and allowing the user to download the CSV file.