# Document Ingestion System

**Client:** Anonymous
**Business Size**: Corporation
**Industry:** Insurance and Risk Management
**Country:** UK
**Technology:** SharePoint, C#, Azure frameworks, MS Logic Apps, Selenium, NUint, Spec Flow

## Objective: Create document ingestion and dissemination system

### The Brief

The objective of this project was to automate the processing of the incoming documents, using one centralized system to receive a document from a client, extract the contents of the document, and forward the content as data to the relevant recipient, which could be any business department in the client's organization. System extensibility was required for futureproofing, and it also had to handle a high daily workload. This enterprise-level application is based on a proof-of-concept exercise undertaken at beginning of 2023.

### Background

Our client has many clients, who provide documents based on particular products for several purposes, e.g. obtaining a quote. These documents can be in a range of formats, e.g. word document, text file, or PDF file. They are then passed to the relevant department for processing and returning any correspondence if required.

### Methodology

A cloud-based workflow solution and hosting system in the client's Azure portal was chosen for this major development.

The technologies adopted included a SharePoint event receiver, Azure Blob (binary large object) Storage, Azure EventGrid, Azure Logic Apps workflow, Azure API management, and Azure Databases.

For the automated testing for this C# application, we used Spec Flow, Selenium, NUnit testing Framework, and Microsoft Logic Apps Framework.

At a high-level, the system is broken down into the following sections:

- The client's SharePoint system provides the front-end Client Portal for document upload. Each document type has their own dedicated SharePoint endpoint.
- A Visual Studio SharePoint event receiver was developed and deployed to process documents uploaded. It sends each uploaded document to Azure Blob Storage, and then creates an Azure EventGrid object to process it in the Azure Logic App workflow. The EventGrid is the trigger to start the workflow. It passes the document and some meta data, such as the name of the file, who sent it, their email, and the API endpoint the document is supposed to go to for processing.
- Next, the document and meta-data is sent to the File Ingestion Engine (FIE) via an API endpoint. The FIE uses the client's Optical Character Recognition software to return the content of the document in the format of Questions and Answers text, along with a confidence level in the legibility of the document. This determines which route in the workflow to take. If FIE has high confidence, then the data is passed to the quote engine where it is processed, and a quote is returned by email. However, if FIE has low confidence, a rejection email is sent with the relevant data. Each workflow run is then saved into the database.

## Challenges

We worked in a team of seven. There were two developers, a project manager, a solution architect, a business analyst, a network operation engineer, and an automated testing solution developer.

There were a number of challenges. With Selenium you can record the actions you take on a website. Selenium IDE then lets you export this recording into a NUnit C# class. Unfortunately, there had to be a lot of refactoring of the generated code as the exported version was very basic, and did not cater for the fact that not all the elements on the web page are rendered straight away. This generated a lot of 'element not found' errors. An additional challenge was that the ID of the elements were generated for a different environment, so more investigation of the code was needed to identify the element in question to be tested. Resolving this issue was time consuming and required careful changes to the code.

Another challenge was SpecFlow and learning how to use this framework. It has its quirks and due to time-constraints, we had to get it working with the existing code without the luxury of refactoring the code to make it more efficient.

Using the Microsoft Framework for Logic Apps was also challenging. While it automatically generated test code, the workflow was not triggered and limited relevant documentation could be found. We suspected the reason for this was SharePoint used by the Azure EventGrid object to upload the blob into the workflow, so it was already being triggered. As a result, the code had to be changed to examine the workflow run history list and identify the document uploaded. Then the remaining tests of each action in the workflow could be performed.

**Consultant Contribution**

Our consultant was responsible for producing a solution for automating end-to-end testing of this system from receipt by SharePoint, through the Azure Logic App workflow, to the response email. They also created high-level documentation covering the scope of the automated testing and developed a Specflow project with two types of Features and their Step Definition classed.

The first development was created to test the Logic App, a blob of the document was created, then an EventGrid object was used to move the blob into the workflow and trigger the processing. When the document reached the Workflow Run History list, it was checked against the expected action result.

The next stage was to run the Selenium code to open SharePoint, upload the document, and obtain the workflow run history of the document. The action results were then checked against the expected results.

**Lessons learned**

Having an involved, friendly, and approachable team was a significant benefit. Our consultant enjoyed the responsibility of owning the automated testing aspect of the project, with the team aware there would be a learning curve for this type of work.