Automated Insurance Analytics

Client: Anonymous
Business Size: Corporation
Industry: Insurance and Risk Management
Country: UK
Technology: C#, ASP.Net Core 6, Blazor, Bootstrap

Objective: Create an Automated Insurance Analytics System

The Brief

The project is an automated insurance analytics system for a major UK reinsurance broker. While this is composed of a suite of semi-independent subsystems, all are needed to meet the overall objectives.

Background

The client is a long-standing international insurance broker that has also become prominent in risk management, and HR & benefits.

There are many vendors who model catastrophes for reinsurance purposes. A catastrophe event (earthquake, fire, flood, landslide, etc.) applies to a geographical location, the building, and the insurance cover of the building. The system models the financial cost of potential claims for such disasters across the world.

Each vendor has their own catastrophe modelling engines and produces their own data, reports, and analytics. The main objective is to compare different vendors' catastrophe models against each other to identify where there may be savings.

In the absence of a standard data format for this information, the client has to import the data from each vendor, transform it to their standard format, and then transform it again to the format of the vendor catastrophe model they want to compare it with, while producing reports and analytics. There are many vendors with different data formats and the size of the data is in petabytes.

The main subsystems are:

- Job Manager – Deals with all the SQL Server jobs that are executed automatically or manually depending on the environment.
- Client Manager – Manages all the clients that are associated with a project, including obtaining data from a source vendor, and transforming and exporting the data to a target vendor's catastrophe modelling engine.
- User Manager – Manages the users of the system i.e. developers, catastrophe modellers, and actuarial professionals.
- Project Manager – Manages the project input submissions, project data, trans-formations, export submissions, analytics, and reports.

Methodology

To develop and enhance the front-end UI using Blazor server-side web pages and components, both Bootstrap and Blazorise component libraries were needed. These allowed for the look and feel of the UI, and the standard set by the team regarding UI development to be followed. The UI had to be adapted to a tablet format as the amount of data that needed to be displayed meant that a typical mobile first design approach was not suitable.

The project was an ASP.Net Core Blazor server-side web application, utilising classes and APIs for CRUD operations, and ADO.Net framework to communicate with the SQL Server database. The innovative part of this project was the creation of the bespoke Data Access layer project and its classes. This was partly due to the requirement that everything developed should not be a black-box and when debugging/maintenance is required, necessary skills are available. The amount of data in this system is in petabytes and control and understanding of the code was paramount.

There were weekly stand-up meetings with a log of the tasks and sub-tasks with estimates of the time expected to complete the tasks. The client's re-insurance department was heavily involved as they held the relevant business knowledge. During these meetings we would discuss our progress and any issues. We would also change priorities accordingly if there were any unexpected delays, which could be the result of testing and fixing or modifying components of the development work. High coding standards were maintained in part with regular code-reviews, and our consultants being allowed a significant degree of ownership of some crucial tasks.

We used C# developed using Visual Studio 2022. The web application was developed using as ASP.Net Core 6.0 Blazor server-side application. For the styling of the UI we used Blazorise and Bootstrap component libraries. We deliberately began with the default settings for all these UI components where possible as we didn’t want to start changing things without user feedback. Azure DevOps was used to maintain our code versions, using development branches and merging when necessary, with Azure Continuous Integration/Continuous Deployment to push the changes into the staging environment for the catastrophe modellers to do their testing.

Challenges

The client engaged OCS Consulting to develop specific functionalities that were planned to be released to a demanding timescale. In addition to the enhancements to the core system, existing in-house tools needed to be further developed, enhanced, or upgraded to help with the setup of lookup data, SQL Scripts, and code generation that are important for efficient operation.

Due to the complexity of the project the team had two of its own developers as well as two developers from OCS Consulting and the outstanding work was split so that the deadline could be met.

The tasks we were given were complex. We had to use existing code representing the tables and lookup data instead of writing our own back-end code for CRUD operations. We needed to understand each of these before starting any coding. With limited assistance available from the client, we examined the code from previous example test pages to get a good idea of the tasks involved. We were expected to investigate as much as possible and then discuss our strategy for resolving the task at hand. It was important to gain business knowledge to ask the right questions when we needed assistance.

Initially it was hard to navigate through the many projects as there were multiple layers to negotiate. Over time we became familiar with each one and understood the code changes required.

In the face of these challenges, our consultant was able to adapt and develop the skills necessary to successfully complete their tasks and contribute to the successful delivery of the project.

Consultant Contribution
As a full stack ASP.Net Core web developer, while our consultant's main role was to develop the Data Format and Transformation functionalities, additional requirements included modifications to two application tools. Previous experience on ASP.Net Core MVC applications made the dependency injection concept easy to understand.

Our consultant created Blazor web pages and components and the code-behind files for each function. Also, because this is a multi-tiered architecture, enhancements were needed to the business and data layer classes. For the database, while no new tables were required, SQL Scripts and stored procedures were created. This was done for:

- DataFormat, describing each data format file, and breaking down the content of each to details such as the name of the file, its location, and what each column of the file is and its data type. This function also includes a component that lists sample files and the functionality to download any of these sample file.
- Transformation, a complex module with multiple sections, including a project header section that breaks down the project's meta data, and a section for listing the Transformation jobs currently running and those already complete. The new Transformation page had several sections again with the project meta data detail, and a list of portfolios (a portfolio as a filtered list of imported submissions from the source vendor). Other sections were used to identify the target transformation and identify the geographical location, the building, and the insurance cost of the building. This was a complex task and a lot

of testing was required to ensure all relevant data was correctly populated.

The Code Generator project (a WinForm project) is a tool to build POCO classes representing any new or updated entity in the SQL Server database. This needed to be rebuilt with DOT.Net 6.0 and upgraded with the latest Nuget Packages, and thoroughly tested before deploying.

The Dev Centre project (a Blazor web project) required creation of a UI and the code-behind classes to obtain the text and related parameters of complex SQL queries stored in database tables in multiple environments. These records were compared to establish if there were any changes using hashbyte functionality for checking the text and simple comparisons for the parameter fields. Once identified, users have the choice to migrate the new or updated versions from the Development environment to the staging environment. ADO.Net Datasets, DataTables and DataAdaptors were used for this operation.

Lessons learned
When working on large enterprise applications, it is important not to get overwhelmed. Understanding the business and systems at a high-level can be very beneficial when working on low-level coding and design tasks. Striking a balance between spending time with busy client staff and investigating on your own is important. When investigating how tools work it is also important to study local code to learn the patterns in use at the client, as there are frequently multiple ways to achieve the same result. It is good to get as much exposure as possible before starting any kind of development work.