**System Architecture**

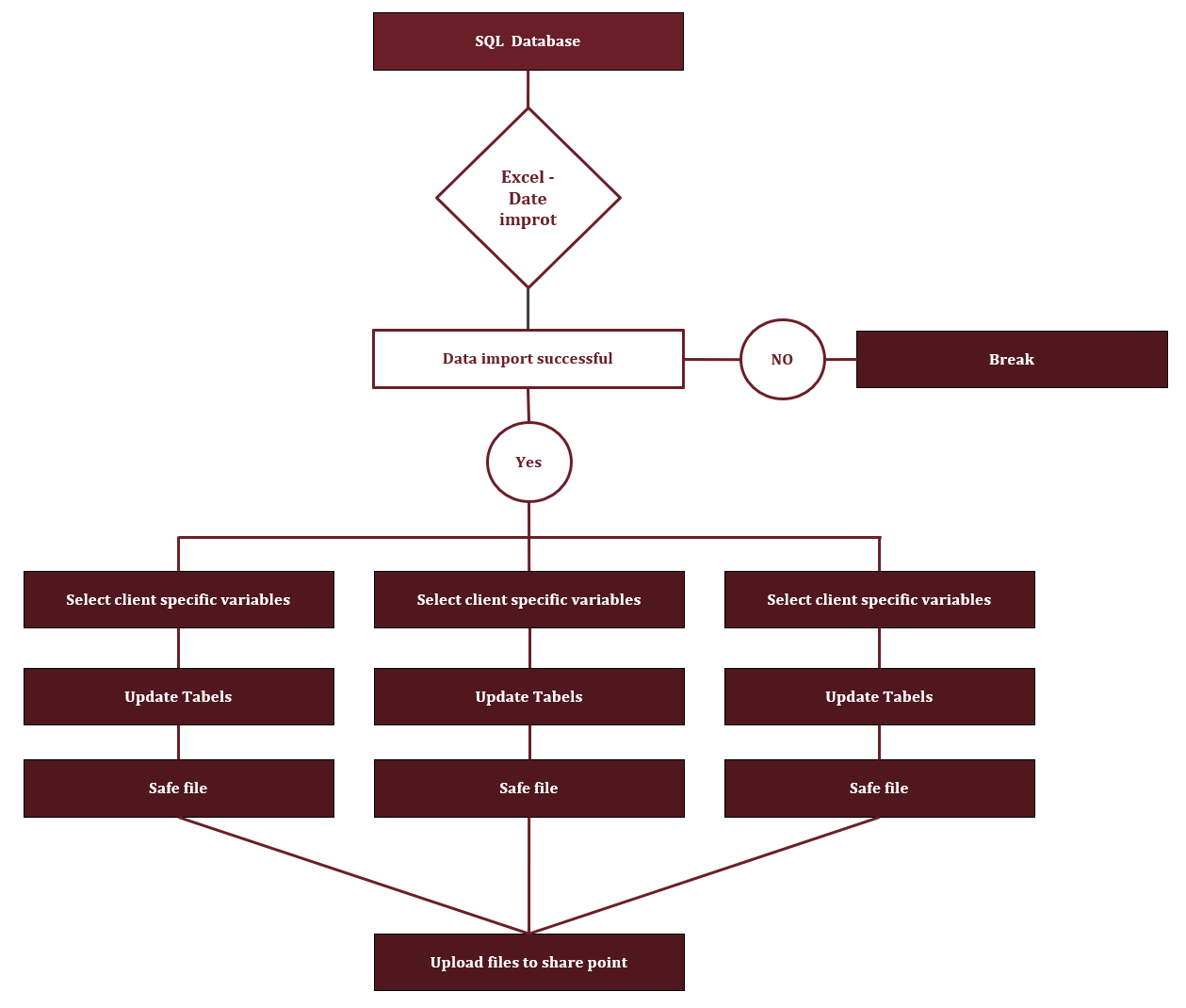
Given the problem description, with its scattered systems and services, the current portfolio builds on a micro service architecture. Fundamentally, micro service architectures enable you to develop based on smaller monofunctional modules (the microservices). The components themselves are developed independently of one another and together form the overall product. This style of architecture has several advantages:

* Independence – flat hirachical development structure; teams work independently and are able to use different programming languages;  
  Robustness – Due to the fact that several teams work independently to develop a unified application, results tend to be more resilient against micro service failure. This is related to a continuous delivery approach, where new developments of a microservice are delivers directly, after thorough testing, regardless of the remaining processes.
* Compatibility – Finally all building blocks are put together. REST-APIs (or similar services) enable an easy exchange of information.
* Scalability – It enables developers to scale individual services according to their needs. This also keeps the end product lean and saves resources.

Taken together, micro service architecture perfectly fit the needs to automate and standardize processes in large corporations. In particular, the flexibility of such a system is a huge advantage.

**Context-View**

The Context-View is concerned with use cases or application scenarios. These describe workflows between components or processes and aim to identify or illustrate architectural elements weak points. They also serve as a starting point for initial architecture tests or implementation drafts. Use case diagram are used for illustration. The following diagram should illustrate the use case for the problem at hand:



**Modul-View**

The Modul-View is concerned with functionality of end-user-systems. Various UML diagrams are used for representation, including class diagrams, communication diagrams, and sequence diagrams as well as component diagram or package diagram. For problem at problem at hand, the end user controls the application via excel (mostly the communication with the SQL Server)

**Runtime-View**

The Runtime-View is concerned with the dynamic aspects a system. It clarifies processes of systems and how they communicate with regards to runtime behavior. This view describes parallelism, distribution, integration, performance and scalability. The corresponding UML diagrams include i.e. activity diagrams. Most critical for the system is a successful database query. The diagram depicted under “Context-View” illustrates the system behavior if the query fails.

**Distribution-View**

The Distribution-View describes systems from system architect’s point of view. This view is concerned with the distribution of software components on a physical level (i.e. the assignment of these hardware parts) and the communication between these components. The associated UML diagram is the distribution diagram. It is safe to assume that data bases, relevant systems as well as desktops with Excel installed are already in place. Thus, this portfolio builds on this infrastructure.