****

**PROJECT: Exam File Manager (EFM)**

**TECHNICAL SPECIFICATIONS DOCUMENT**

Created: 11/10/2021

**Document Revision History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Release No.** | **Date** | **Revision Description/**  **Requirement #** | **Modified By** | **Approved By** | **Approval Date** |
| V1.0 | 11/11/2021 | Revision | Norman Avery |  |  |
| V2.0 | 11/17/2021 | Revision | Norman Avery |  |  |
| V3.0 | 11/23/2021 | Revision | Norman Avery |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

Table of Contents

[1.0 Proposed (Basic) User Interface Design for Exam File Manager 4](#_Toc88589186)

[2.0 GENERAL INFORMATION 5](#_Toc88589187)

[2.2.2 Project Objective 5](#_Toc88589188)

[2.2.3 Overview 5](#_Toc88589189)

[3.0 Architecture 5](#_Toc88589190)

[3.2 Systems Architecture 5](#_Toc88589191)

[3.2.2 High Level Design Describing the Different Type of Systems Interacting with EFM. 5](#_Toc88589192)

[3.2.3 High Level Design Describing the Assembly Plugin and AllPlugins\_Service Object 6](#_Toc88589193)

[3.3 Application Architecture 7](#_Toc88589194)

[3.3.2 High Level Design to Implement and Load the Assembly Plugin 7](#_Toc88589195)

[3.3.3 High Level Design to Configure Services 7](#_Toc88589196)

[3.3.4 High Level Design to Implement Business Logic for Common Features 8](#_Toc88589197)

[3.3.5 High Level Design to Retrieve Lines of Businesses via “RetrieveLinesOfBusiness” method 9](#_Toc88589198)

[3.3.6 High Level Design to Execute Plugin Feature (Business Logic) via “ExecuteBusinessLogic” 10](#_Toc88589199)

[3.3.7 High Level Design to Execute Plugin Feature “Confirm” 11](#_Toc88589200)

[3.3.8 High Level Design to Execute Plugin Feature “Delete” 12](#_Toc88589201)

[3.3.9 High Level Design to Execute Plugin Feature “Import” 13](#_Toc88589202)

[3.3.10 High Level Design to Execute Plugin Feature “Index” 14](#_Toc88589203)

[3.3.11 High Level Design to Execute Plugin Feature “Repair” 15](#_Toc88589204)

[3.3.12 High Level Design to Execute Plugin Feature “Scan” 16](#_Toc88589205)

[3.3.13 High Level Design to Execute Plugin Feature “Search” 17](#_Toc88589206)

[3.3.14 High Level Design to Execute Plugin Feature “View” 18](#_Toc88589207)

# Proposed (Basic) User Interface Design for Exam File Manager

Graphical user interface

Description automatically generated

# GENERAL INFORMATION

### Project Objective

The current Exam File Manager application needs to be redesigned for extensibility. This new design will enable the ability to dynamically configure the application, customized for a specific line of business. The method of configuration employs the use of a developer plugin (assemblies) that will configure the application at run time. To make it possible for the web application to interact with external systems, additional services will be configured by the plugin, and injected into the application via dependency injection.

### Overview

Currently, the EFM application uses excessive if-then logic to distinguish which line of businesses that the application’s business logic executes against. The new approach will increase scalability, as Lines of Business are installed to the web application by assembly plugins, being authenticated based on user roles and plugin configuration. The underlying web application will be generic in nature, but configurable and extensible by the use of assembly plugins.

# Architecture

## Systems Architecture

### High Level Design Describing the Different Type of Systems Interacting with EFM.

Systems that will be interacting with the web application include External File Repositories, a Database, Assembly Plugin, and Local File Systems

Diagram

Description automatically generated

Diagram

Description automatically generated

### High Level Design Describing the Assembly Plugin and AllPlugins\_Service Object

Lines of Business will be installed by implementing an assembly a plugin that contains all necessary information related to a line of business. The plugins will be loaded at runtime and added as a collection to a service object called AllPlugins\_Service Object. This service object will be injected into a web controller via dependency injection, and will be referenced by the web controller to perform all necessary functions related to Exam File Manager, respective to a line of Business.

Diagram

Description automatically generated

## Application Architecture

### High Level Design to Implement and Load the Assembly Plugin

**Current flow**

Currently, EFM does not use custom assemblies to configure the web app specific to a line of business.

**New/Proposed Flow**

A library assembly interface will contain all the required class members and function declarations that will be used to extend common functionality.

### High Level Design to Configure Services

**New/Proposed Flow**

The assembly’s implemented plugin class will contain lists of key-value pairs used to create service objects that are utilized by the application to connect to a file repository and database.

These respective objects will be created and injected as singleton service objects at runtime. Additionally, the assembly plugin object itself will be injected as a singleton service, which will be used to extend the business logic for common features. These objects will be referenced by the web application’s controllers via dependency injection.

Diagram

Description automatically generated



Diagram

Description automatically generated

### High Level Design to Implement Business Logic for Common Features

**Current flow**

Currently, the application uses excessive if/then statements to execute business logic specific to a line of business.

**New/Proposed Flow**

The assembly plugin will contain a list that defines if common features will be available for a specific line of business. Additionally, the plugin will include methods that are extended from an interface, containing custom business logic. At runtime, the “PluginLoaderHelper” class will load the plugin, and instantiate an object containing the respective business logic for each common feature. The web application’s Controller will reference these methods based on parameters passed throughout the web application’s HTTP requests. The following sections demonstrate how the web application’s Controller will interact with the plugin’ respective methods, to execute the plugin’s business logic.

### High Level Design to Retrieve Lines of Businesses via “RetrieveLinesOfBusiness” method

**Current flow**

Currently, the application uses excessive if/then statements to find all lines of businesses that a user belongs to.

**New/Proposed Flow**

For demonstration purposes, a GUI (Graphical User Interface) landing page will be the first point of entry to the Exam File Manager. Before generating the landing page, the web controller will utilize the authentication module to retrieve the roles and respective security levels that a user belongs to. The controller will then reference the AllPlugins\_Service\_Object (injected as a service object via dependency injection by the application’s startup class) to find a complete list of all “Lines of Business” that has been installed in the web application via assembly plugins. If no Lines of Business are found, an error message will display.

The controller then will call a function belonging to the AllPlugins\_Service\_Object, namely “RetrieveFeatureNames” – to retrieve a list of available features defined in the plugin. The installed Line(s) of Business will then be sent to the landing page with a list of respective features that are enabled for each line of business. The user can then select a line of business, which will be sent back to the controller for further processing and rendering of a dynamic razor page.

Diagram

Description automatically generated

### High Level Design to Execute Plugin Feature (Business Logic) via “ExecuteBusinessLogic”

**Current flow**

Currently, the application uses excessive if/then statements to execute business logic, using security based on a user, permissions, and line of business.

**New/Proposed Flow**

Execution of business logic will no longer need to apply if/then statements to filter executed code specific to a Line of Business. Instead, at this point, the Exam File Manager Environment will already be filtered, and only applicable to a single, specific Line of Business, by the previous actions taken in 2.2.4-2.2.5. Business Logic code execution will proceed after a user click event respective to the feature (ex. Import button clicked). The action will then send the following parameters to a controller - role, security levels, LOB, feature name, and data (params) respective to the feature. The controller then references the AllPlugins\_Service\_Object to validate the availability of the feature. Once validated, the business logic associated with a feature is executed by calling the function “ExecuteBusinessLogic”. Next, a razor web page view is rendered which will return the user to the Exam File Manager.

Diagram

Description automatically generated

### High Level Design to Execute Plugin Feature “Confirm”

**New/Proposed Flow**

The ‘Confirm’ button exists to confirm that the indexing of a Medical Record is complete. After the user has indexed a Medical Record, the ‘Confirm’ button will be enabled. When the ‘Confirm’ button is clicked, it will be disabled and greyed out.

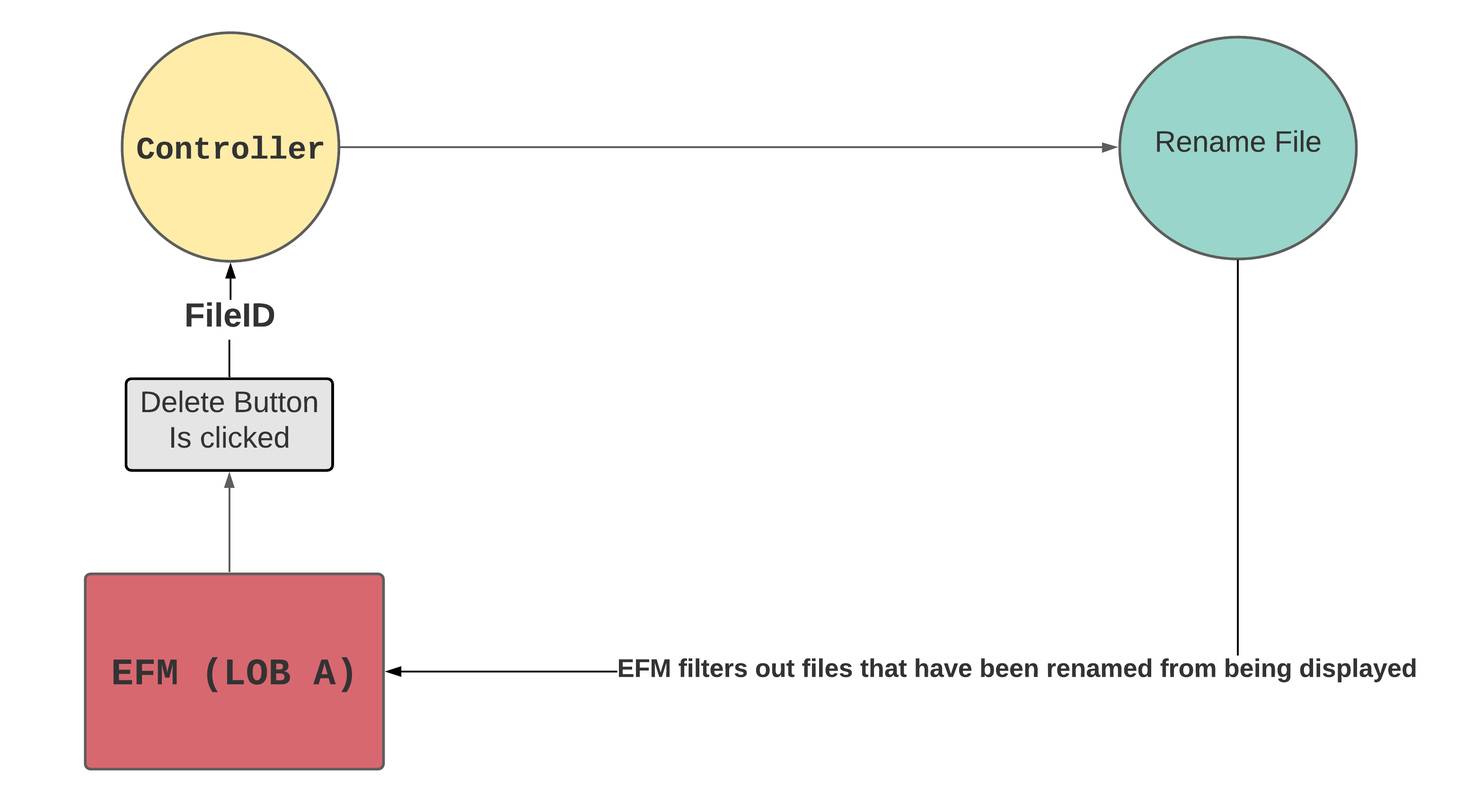
Diagram

Description automatically generated

### High Level Design to Execute Plugin Feature “Delete”

**New/Proposed Flow**

Effectively deletes a file by renaming it to something that will not be displayed.



### High Level Design to Execute Plugin Feature “Import”

**New/Proposed Flow**

The main goal of the import feature is to allow a user from any line of business to import a file. When the file is imported, the controller checks the file’s name. Then it names it according to the user who imported the file.

Diagram

Description automatically generated

### High Level Design to Execute Plugin Feature “Index”

**New/Proposed Flow**

The index button will bring up a window that will require input from the user. The index will set which pages belong to which specialty. For example, pages 1 - 10 would belong to one specialty, and 11 - 13 would belong to another. While indexing, the controller will look for loaded plugins and add it into a global model. The global model will dynamically load tabs onto the layout page where the user can go to different pages based on the loaded plugin.Diagram

Description automatically generated

### High Level Design to Execute Plugin Feature “Repair”

**New/Proposed Flow**

Repair will have two outputs; it will either be a returnable file to be viewed by the user, or an alert notifying the user that the file must be downloaded. The controller first tries to repair the file using NLP. If the file is too corrupted to repair, the controller will notify the user to download the file. The reason why the controller initially does not download the file is because the time to download is costly, thus making it more ideal to repair.

Diagram

Description automatically generated

### High Level Design to Execute Plugin Feature “Scan”

**New/Proposed Flow**

The scan feature allows a physical document(s) to be scanned by a hardware scanner, and then saved into EFM. Clicking the "Scan" button will trigger a button action that calls the controller. The event handler will then invoke the twain drivers on the local computer, to utilize the connected scanner. Finally, the scanned document is saved into EFM.

Diagram

Description automatically generated

### High Level Design to Execute Plugin Feature “Search”

**New/Proposed Flow**

Search allows a user to return a view with specific files associated with two user inputs: an account number and a date range. User enters either date range or account number into a razor form (account number only supports alphanumeric characters), then once the search button is clicked, date range or account number is sent to a controller. The controller generates a query that searches tables in the backend for the files associated to the inputs. If the query is valid, it returns a view with files associated to valid query. Invalid queries return the user to the starting razor form view.

Diagram

Description automatically generated

### High Level Design to Execute Plugin Feature “View”

**New/Proposed Flow**

The purpose of the view feature is to allow the user to view the file associated with a row that is displayed on a web page, respective to a database record. Clicking the “View” button executes the view function. This action sends a request containing the file ID to the controller. The controller then retrieves the file from a repository. Once the file is retrieved, the file is opened and displayed in a new window. If the file is not found, a new window will display the error message “File Not Found”.

