**Spectrum Analyzer Tool**

**Design Document**

# 

Chris Williams, George Vendeta, Seema Kumaran, Jeffrey Amakihe, Alexandro Pinion

Date: 10/25/2023

Department: Computing and Software Engineering

Course: SWE 7903 - Capstone

Professor: Dr. Parizi

**Table of Contents**

[1. INTRODUCTION 3](#_Toc149170358)

[1.1 Purpose 3](#_Toc149170359)

[1.2 Scope 3](#_Toc149170360)

[1.3 Overview 3](#_Toc149170361)

[1.4 Reference Material 3](#_Toc149170362)

[1.5 Definitions and Acronyms 4](#_Toc149170363)

[2. SYSTEM OVERVIEW 5](#_Toc149170364)

[3. SYSTEM ARCHITECTURE 6](#_Toc149170365)

[3.1 Architectural Design 6](#_Toc149170366)

[3.2 Design Rationale 9](#_Toc149170367)

[4. DETAILED DESIGN 11](#_Toc149170368)

[4.1 UI Component 11](#_Toc149170369)

[4.2 UI Layout 13](#_Toc149170370)

[4.3 Processor Component 15](#_Toc149170371)

[4.4 Data Manager Component 17](#_Toc149170372)

[5. DATA DESIGN 19](#_Toc149170373)

[6. HUMAN INTERFACE DESIGN 20](#_Toc149170374)

[6.1 UI design 20](#_Toc149170375)

[6.2 UX design 26](#_Toc149170376)

[7. APPENDICES 27](#_Toc149170377)

**Table of Figures**

[Figure 1 - Architectural Diagram of Spectrum Analyzer Tool 8](#_Toc149170378)

[Figure 2 – Basic UI Component Class Diagram 12](#_Toc149170379)

[Figure 3 - UI Layout Class Diagram 14](#_Toc149170380)

[Figure 4 - UI Layout Sequence Diagram 15](#_Toc149170381)

[Figure 5 - Processor Component Class Diagram 16](#_Toc149170382)

[Figure 6 – Processor Component Psuedocode 17](#_Toc149170383)

[Figure 7 - Data Component Class Diagram 19](#_Toc149170384)

[Figure 8 – Main Window Example 20](#_Toc149170385)

[Figure 9 - Runtime Calibration Window 21](#_Toc149170386)

[Figure 10 – Processing Bar 22](#_Toc149170387)

[Figure 11 - Error Prompt 24](#_Toc149170388)

[Figure 12 - Video Select Prompt 25](#_Toc149170389)

[Figure 13 - UX Design Diagram 26](#_Toc149170390)

**Table of Tables**

[Table 1 - Acronyms 4](#_Toc149170391)

### INTRODUCTION

## Purpose

The purpose of this System Design Document (SDD) is to provide an extensive blueprint that describes the architecture and design of a software. Its main goal is to give a precise and comprehensive explanation of the software's architecture, design, component interactions, and means of meeting the requirements. The SDD fulfills several crucial functions. It serves as a communication tool to assist developers, testers, project managers, and other team members in comprehending the architecture of the software system. Throughout the software development lifecycle, the development team uses it as a reference document. It facilitates the documentation of design choices and justifications. The intended audience for this SDD is for developers, Robins AFB stakeholders, testers and project managers.

## Scope

The goal of this Software project is to create, test, and implement a user-friendly program that the Robins AFB can use to solve video analysis and spectrum analyzer difficulties. An independent deployment environment for the client's platform, formatted CSV data representing spectrum analyzer signals, and image signal processing of video are some of the key features. The project's goal is to provide a reliable, scalable solution that addresses the client's issue within the allocated spending limit and time frame.

## Overview

This document will go over the system architecture, data flow, user interfaces, algorithms, data structures, security considerations, error handling, and other subjects are covered in detail in this document. It serves as an essential means of communication for development teams, testers, project managers, and stakeholders, guaranteeing that all parties participating in the project comprehend the functioning and design of the product. An integral part of the software development lifecycle, the SDD also acts as a point of reference for scalability and future maintenance.

## Reference Material

C. Williams, “Spectrum Analyzer Software Requirements Document (Draft -SWE 7903 Spectrum Analyzer Capstone -Team 4),” Sep. 2023.

## Definitions and Acronyms

Table - Acronyms

|  |  |
| --- | --- |
| **Name** | **Acronym** |
| Air Force Base | AFB |
| Comma Separated Value | CSV |
| Electronic Warfare | EW |
| Model View Controller | MVC |
| User Interface | UI |
| User Experience | UX |

### SYSTEM OVERVIEW

As a key aviation hub, Robins Air Force Base (AFB) uses outdated Spectrum Analyzers to measure the quality of transmission signals within their ground stations. These Analyzers gather information while in flight, producing hours' worth of crucial data that is necessary to preserve the integrity of communications networks. However, there are operational inefficiencies and a high human resource cost associated with the current manual procedure of examining video recordings of these datasets for anomalies in signal quality. In response to this challenge, KSU's Master of Software Engineering Capstone students have been requested by Robins AFB to develop an automated system for the analysis of flight data. This will guarantee prompt identification of problems with signal quality while optimizing operations and resource use. This group of students behind this project will create an open-source standalone application that will operate on the computers that are intended for use by Robins AFB personnel. The application will offer a guided process for loading relevant data, processing it, and producing output resources that will be most helpful to the stakeholders at Robins AFB.

### SYSTEM ARCHITECTURE

## Architectural Design

This section includes a high-level structure design that outlines the main elements, how they relate to one another, and how the application is organized overall. It acts as a construction blueprint for the software, assisting developers in comprehending how different system components interact to produce the intended functionality. Important non-functional needs like performance, maintainability, and scalability are covered. This section will also provide a high-level road map that directs the development team in creating a system that satisfies the project's functional and quality objectives while taking flexibility and maintenance ease into account is the aim of architectural design. For the Spectrum Analyzer Tool application to remain viable and successful over the long term, a well-designed architectural design is essential. Below are the main components of the systems architecture:

* **User Interface (UI) Layer**:
  + The desktop application's user interface is known as the UI.
  + It offers an easy-to-use interface so people can engage with the program.
  + It has options for image processing, video data display, and file input for choosing video files.
  + Additionally, the UI provides controls for exporting CSV files and starting the data conversion process.
* **Logic Layer**:
  + Video Processor:
    - This part extracts pertinent data by processing the loaded video frames.
    - It could involve filtering, feature extraction, picture analysis, and other image processing methods.
    - The data has been processed and is ready to be converted to CSV format.
  + This layer will perform the business logic for the application.
* **Data Management Layer:**
  + CSV Data Converter:
    - This module creates a CSV file from the processed video data.
    - It arranges the information in a tabular style that can be exported as CSV.
    - The CSV format may be configured by users with options to set headers and delimiters, for example.
  + This layer will be responsible for creating all of the non-volatile data storage for the application.
* **Exceptions and Logging Layer**:
  + Put in place an error-handling system to handle unforeseen problems that arise when loading, processing, or converting videos.
  + Produce logs for reporting errors and debugging.
* **Distribution Layer**:
  + Builds all packages and bundles all preset libraries, dependencies, and components that makes it simpler to deploy the application to the user.
  + Generates a standalone executable that operates independently on a computer system without the requirement for extra resources or dependencies.

The Figure 1 below will showcase an overview diagram of the architectural design for the Spectrum Analyzer Tool and how each of the components in the application provide key functionality. Each of the diagram’s components are described above.

A diagram of a software application

Description automatically generated

Figure - Architectural Diagram of Spectrum Analyzer Tool

## Design Rationale

The primary software architecture pattern chosen was the Model View Controller (MVC) pattern. The key benefit of implementing the MVC architecture in a desktop application is that it encourages a distinct division of responsibilities, which improves the program's maintainability, scalability, and flexibility. It is possible to modify the user interface (View) without also changing the data processing logic (Model) that underlies it, and vice versa. Because developers can work on different components independently, this separation also makes testing and cooperation easier.

**View** - The choice for the user interface is crucial because it acts as the main point of contact between the user and the program, enabling clear and effective communication and will act as the View in our architectural pattern of choice. A well-thought-out user interface improves the user experience by offering a visual framework that leads users through the functionalities of the application, facilitating simpler navigation, comprehension, and usage of the capabilities. It guarantees that users may engage with the program in a way that is both aesthetically pleasing and easy to use, which eventually enhances productivity, usability, and user pleasure.

**Controller** - An image processor was added because it is beneficial to increase the capability and versatility of a desktop program for manipulating video files by adding image processing. Through image processing, the application can extract useful data from video frames, including motion tracking, object recognition, and facial detection. Consequently, this enables users to carry out diverse tasks such as data analytics, video-image manipulation, and surveillance with increased accuracy and automation. Additionally, image processing can enable features like visual effects, noise reduction, and video stabilization, all of which enhance the output's overall quality and aesthetics. It gives the application an extra layer of intelligence, which makes it a potent tool for the specific use-case intended for the Robins customers.

**Model** - For effective data organization, storage, and retrieval, a desktop program processing photos and converting them to CSV data must have a data management layer. This layer gives the software the ability to organize and handle the translated image data in a methodical manner, facilitating user access and searching. Version control, indexing, and metadata tagging are all made possible by it, and these features are very helpful for complex image processing jobs. It also makes it easier to manipulate, transform, and export data into CSV format, which improves the application's ability to handle data and meets the primary requirements of the developed system. The application is a useful tool for a variety of picture data analysis and reporting jobs because a strong data management layer ensures that users can work with structured and relevant data while streamlining the conversion process and enhancing overall data quality – this will afford the Robins customers more leverage with the different data analytics that they will perform on the applications CSV data.

In addition to the Model-View-Controller (MVC) pattern, there are two other architectural patterns that were considered: the Single Responsibility Principle (SRP) and the Model-View-ViewModel (MVVM). Because MVVM uses a ViewModel to segregate the View from the underlying data, it makes data binding simpler and allows for dynamic UI updates. This makes it ideal for applications with sophisticated user interfaces, such image processing apps. However, we did not need the added sophistication to our user interface and preferred the MVC to allow us the benefits of having a designated controller – the controller was more beneficial for our requirements of image processing opposed to the MVVM architecture overall. Based on architectural design principles, SRP promotes modular and maintainable code by highlighting the Single Responsibility Principle. It may not give as clear a boundary between the View and Model as MVC or MVVM, which could be a disadvantage in our desktop application being developed, even though it can result in simpler code and higher testability. On that notion, choosing the MVC seemed better suitable for our system since our customers may build upon our code base and add features to the application after our delivery.

### DETAILED DESIGN

As referenced in section 3.1 and depicted in Figure 1, our system design is comprised of a few core components. This section will explain the purpose of each component and how they are broken down with an object-oriented approach for developing the design of the system. Our design leverages an object-oriented design to help create organized, modular and reusable code. It mirrors real-world structures and relationships, making it easier for developers and our Robins stakeholders to understand and maintain the code during development, and after delivery. We will use class diagrams to represent the objects, attributes, methods and classes used to design the application. The components described will be: (1) UI Component, (2) Processor Component, (3) Data Component, and (4) Exception and Logging Component.

## UI Component

The user interface is a critical component of our chosen design since it serves as the primary point of contact between the user and the program, facilitating efficient and transparent communication. A well-designed user interface enhances the user experience by providing a visual framework that guides users through the application's operations, making it easier for them to navigate, understand, and utilize the capabilities. The UI component was broken up into the following:

* **Class**: The central element of a class diagram is a rectangle representing a class, with three compartments:
* **Class Name**: At the top of the rectangle, you specify the name of the class.
* **Attributes**: In the second compartment, you list the class's attributes (data members or fields).
* **Methods**: In the third compartment, you list the class's methods (functions or behaviors).
* **Associations**: Lines connecting classes to indicate relationships. These lines may have labels to describe the type of relationship, like "has-a," "uses," "is composed of," etc.
* **Inheritance/Generalization**: A solid line with a hollow arrowhead connecting a subclass to a superclass. It represents an "is-a" or "is a kind of" relationship.
* **Realization/Interface**: A dashed line with an arrowhead connecting a class to an interface or an abstract class. It indicates that the class implements the interface.
* **Aggregation**: A diamond shape on one end of the association line, pointing to the "whole," representing a "part-whole" relationship. It implies a weaker form of ownership.
* **Composition**: A filled diamond shape on one end of the association line, indicating a strong "whole part" relationship. The "whole" manages the lifecycle of the "part."

The UI component ensures that users can interact with the application in a way that is easy to use and aesthetically beautiful, which subsequently improves user satisfaction, productivity, and usability. Below we will show the main objects, attributes and methods of the UI component shown in a class diagram.

A screenshot of a diagram

Description automatically generated

Figure – Basic UI Component Class Diagram

### UI Layout

For the spectrum analyzer, a user is provisioned with a simple UI, where user can select the desired video file for processing, select the calibration values for processing, and save the output csv file in the desired location.

* **File Uploader Window**:
  + In this screen, a user can select the input file location where the video file is available. It can be in mpeg format. The location where the user wants to save the output csv file can also be selected from this UI. Once the location is finalized, user can navigate to next screen.
* **Calibration Window**:
  + This screen allows users to proceed with the default High and Low values of RGB for the video processing. If the user wants to change the RGB values of High parameters, user can click on 'Edit' button and select the color from the color picker. The UI get reflected with the RGB values of selected color.
  + User can edit RGB values of Low parameters as well.
* **Video Processing Window**:
  + This window waits for completion of processing and progress percentage being showed to customer real time. Once the processing is over, the user can stick on the location selected for the output or he/she can change the location by going back to the initial screen. Once the location is finalized, the output file getting saved in the selected location. Now the user can close the window.

In UI Layout class diagram below, the main class is "Spectrum Video Analyzer," which has instances of the FileUploaderWindow, CalibrationWindow, and VideoProcessingWindow classes. FileUploaderWindow allows the user to select and upload files. CalibrationWindow handles calibration and stores calibration data in a CalibrationData object. VideoProcessingWindow processes video and displays results using a Video class and SpectrumData. The CalibrationData class holds calibration parameters, and the Video class represents the video being processed. The SpectrumData class holds the data related to spectrum analysis.

A diagram of a computer

Description automatically generated

Figure - UI Layout Class Diagram

The sequence diagram below will display the interactions between user event processes organized in a timeline. The objects, procedures, and messages that are exchanged to perform the functionality of the user interface will be displayed in Figure 4.  
A diagram of a software project

Description automatically generated

Figure - UI Layout Sequence Diagram

## Processor Component

The addition of an image processor was made since it improves the functionality and adaptability of a desktop application for working with video files. The program can extract relevant information from video frames, such as motion tracking, object recognition, and facial detection, by using image processing. As a result, users can do a variety of tasks with more accuracy and automation, including data analytics, video-image editing, and surveillance. Furthermore, image processing can enable elements that improve the overall quality and aesthetics of the output, such as visual effects, noise reduction, and video stabilization. It adds another level of intelligence to the application, making it a powerful tool for the use-case meant for Robins clients. Below we will show the main objects, attributes and methods of the Image Processor component shown in a class diagram.

A computer code with text

Description automatically generated with medium confidence

Figure - Processor Component Class Diagram

To provide an easy-to-understand, language-neutral method for outlining and organizing a program's logic, assisting in the comprehension, creation, and resolution of challenging issues in software development and image processing done by the processor component, below showcases pseudocode of the algorithms used to process each image.

# Import necessary libraries or modules for image processing and CSV handling.

# Open the video file for reading.

video = open\_video("video\_file.mp4")

# Open a CSV file for writing the results.

csv\_file = open\_csv("output.csv")

# Iterate through each frame in the video.

for frame in video.frames:

# Process the frame to find specific colors and objects.

processed\_frame = process\_frame(frame)

# Extract numerical values for the colors and objects found.

colors\_data = extract\_colors(processed\_frame)

objects\_data = extract\_objects(processed\_frame)

# Write the data to the CSV file.

csv\_file.write\_row(colors\_data + objects\_data)

# Close the video file and CSV file.

close\_video(video)

close\_csv(csv\_file)

# Define functions to process frames, extract colors, and extract objects.

function process\_frame(frame):

# Implement image processing techniques on the frame.

# This might include color detection, object recognition, etc.

processed\_frame = apply\_image\_processing(frame)

return processed\_frame

function extract\_colors(frame):

# Extract specific color information from the processed frame.

# Convert the color information to numerical values.

colors\_data = convert\_colors\_to\_numerical\_values(frame)

return colors\_data

function extract\_objects(frame):

# Extract information about objects from the processed frame.

# Convert object information to numerical values.

objects\_data = convert\_objects\_to\_numerical\_values(frame)

return objects\_data

# Additional functions for working with video and CSV files.

function open\_video(file\_path):

# Open the video file and return a video object.

return video

function open\_csv(file\_path):

# Open a CSV file for writing and return a CSV object.

return csv\_file

function close\_video(video):

# Close the video file.

close(video)

function close\_csv(csv\_file):

# Close the CSV file.

close(csv\_file)

Figure – Processor Component Psuedocode

## Data Manager Component

The application will process images and convert them to CSV – this requires that the data has a data management component for efficient data organization, storage, and retrieval. This component allows the software to systematically process and arrange the translated visual data, making it easier for users to access and search. It enables capabilities like version control, indexing, and metadata tagging, all of which are particularly useful for intricate image processing tasks. Additionally, it facilitates data manipulation, transformation, and export into CSV format, which enhances the application's data handling capabilities and satisfies the main needs of the system that was designed. The data manager component is useful for a variety of picture data analysis and reporting jobs because it ensures that users can work with structured and relevant data while streamlining the conversion process and enhancing overall data quality. Below we will show the main objects, attributes and methods of the Data Manager component shown in a class diagram.

### DATA DESIGN

A diagram of a component

Description automatically generatedThis section will show the basic data design in organizing and storing data in a structured manner. It entails outlining the kinds of data (text, numbers, dates, etc.) that the application will utilize as well as how those data interact with one another. To store and retrieve data efficiently, this design may entail building tables, fields, and relationships in a data structure. It also entails making sure that the data structure satisfies the needs and functionalities of the application and facilitating the efficient management and manipulation of data by the program. The Signal, Data and CSV classes of the application will be depicted to explain the data design. We have multiple functions which retrieve float data types of all the pertinent information within the Signal class. This information is then stored in an array list. The signal list is then called to the Data class, where a CSV object is then created. The CSV class then receives and stores the signal list which contains all the Amplitudes.

Figure 7 - Data Component Class Diagram

The data design represented by the class diagram above serves the purpose of managing massive volumes of image data and it guarantees the effective storage and retrieval of the processed data. Effective organization and classification of object data by the application, which makes it available for upcoming analysis or system integration, is another benefit of proper data design. Furthermore, it makes the application more flexible and compatible with a wider range of data processing workflows by streamlining data interchange and facilitating collaboration with external tools or databases by organizing data for export to a CSV file.

### HUMAN INTERFACE DESIGN

This section will showcase the process of developing the interactive and visual components of software application, along with any tangible gadgets that allow users to engage with the UI design. To improve the user experience, it focuses on creating visually appealing interfaces, simple navigation, and user-friendly layouts. To increase user pleasure and productivity, human interface design aims to provide interfaces that are intuitive, effective, and aesthetically beautiful.

## UI design

Temp This section includes developing the interactive and visual components of the system and showcases the product that people interact with directly. To provide a graphical user interface that is both aesthetically pleasing and easy to use, this section will focus on the arrangement, style, and functioning of the interface, including components like buttons, menus, screens, and icons. To create a smooth and intuitive user experience, user interface (UI) designers prioritize efficiency, consistency, and clarity when designing digital or physical interfaces. This improves the user's ability to interact with and navigate the system or product. The figures below will display the various visual components that depict the UI design.

A screenshot of a computer

Description automatically generated

Figure – Main Window Example

A screenshot of a computer

Description automatically generatedThe calibration window allows the user to select which portion of the screen that the user wants the application to grab signal data from. The user can also select the specific values for a given signals RGB value to improve the quality of the image processing. Once the calibration parameters have been set by the user, they can create an event to start processing by clicking the Go! Button.

Figure - Runtime Calibration Window

A screen shot of a computer

Description automatically generated

Figure – Processing Bar

A screen shot of a computer

Description automatically generated

Figure 10 above displays a progress bar that is used as a common visual feature for the user and shows users where a task or process—like the processing of images within a given video file—is in relation to its completion. Users can clearly see how much of the task has been performed and how much is left with the help of a steadily filling bar or indicator.

A screenshot of a error message

Description automatically generated

Figure - Error Prompt

The error prompt will handle various exceptions thrown by the application. This will allow the user to be notified of any important errors that may occur while utilizing the application.

A screenshot of a computer

Description automatically generated

Figure - Video Select Prompt

The video select prompt will guide the user in navigating, selecting and processing which user desired video file will be processed, along with choosing the output directory of the CSV data generated by the application.

## UX design

In this section, we will look at the procedure for developing and improving a user's entire experience with a system, product, or service. To ensure that the user's journey is simple, effective, and fulfilling, it focuses on first understanding user behaviors, needs, and preferences, then carefully creating the user interface and interaction flow. In the end, UX designers want to develop a product that not only satisfies user expectations but also delights and achieves their goals. They optimize every aspect of the user's interaction, from performance and accessibility to visual aesthetics and ease of navigation. Below in Figure 13 showcases the UX design:

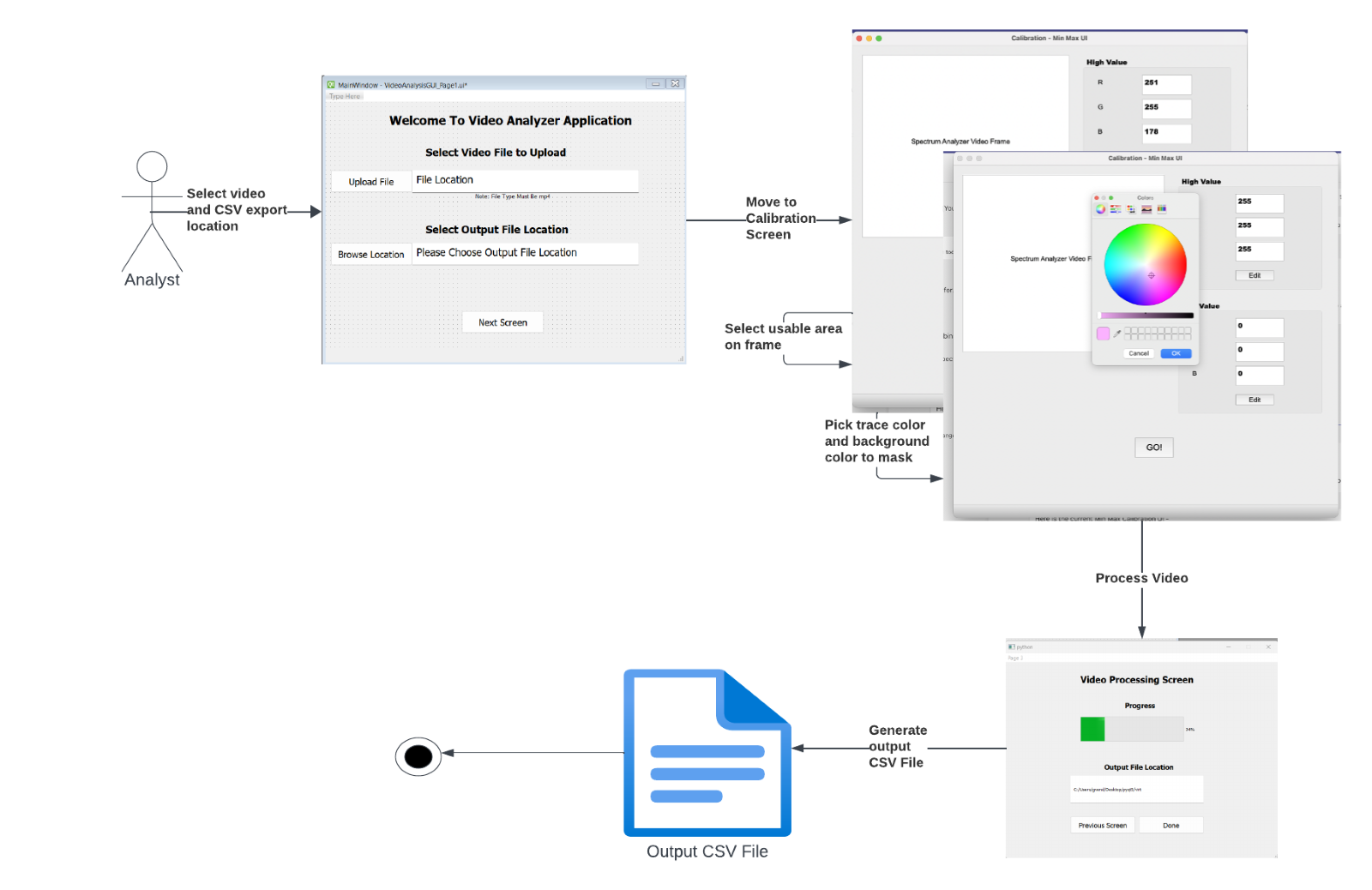


Figure - UX Design Diagram

### APPENDICES

|  |  |  |  |
| --- | --- | --- | --- |
| **Req Id** | **Requirement** | **Design Feature** | **Section** |
| 3.1.1,  4.1 | Input data requirements – Required | CLI and UI design | **3.1** |
| 3.1.2.1, 4.3 | Required Output – csv output | Data Management Layer | **3.1** |
| 3.1.2.2 | Optional Output – csv | User Interface Representation | **3.1** |
| 4.2 | Analyze video of signal intercepts | Logic Layer | **3.1** |