APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 7 No. 1 (2026) p. 1015-1036
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

SpliceFound: A Web-Based Image Splicing Detection Tool Using Error Level Analysis (ELA)

Nur Alyaa Nori Sham[1], Nor Bakiah Abd Warif[1]*

[1] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: norbakiah@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2026.07.01.055

Article Info

Received: 20 July 2025
Accepted: 15 June 2026
Available online: 30 June 2026

Keywords

ImageForensics, TamperingDetection, ErrorLevelAnalysis, SplicingAnalysis

Abstract

Social media and digital communication platforms have made online image sharing a commonplace. However, users find it harder and harder to believe that images are legitimate because of the modified information. Many image forgery detection systems currently in use are difficult to use. Their outputs are frequently complicated, not showing clear visual cues, and need advanced interpretation skills. Additionally, they overlook crucial elements like user accounts for recording past analysis or downloadable results. Additionally, non-technical users are demotivated by complex setups. SpliceFound was created as a forensic picture analysis tool to identify tampered areas, with an emphasis on image splicing forgery, in order to address these issues. SpliceFound is a web-based tool that integrates Error Level Analysis (ELA) with thresholding and image masking to provide tampered regions while allowing user account creation and providing additional report on the analysis result. Agile methodology is conducted in the project which emphasizes collaboration, feedback, and adaptability for a user-centered product by the phases of planning, designing, developing, testing, and delivering in iterative cycles. The tool is developed using Python for backend, HTML, CSS with JavaScript for frontend and flask framework. SpliceFound aims to offer a user-friendly image splicing detection tool that provides reasonably accurate results, automated tampering result reports, and accessible features for forensic practitioners, legal officers and non-technical users. Additionally, it offers the ability to track the history of previous analysis.

1. Introduction

Digital forensics is the practice of finding, preserving, and analyzing digital evidence to help in investigations. It is a very important matter in solving crimes, ensuring that justice is served, and keeping digital information safe in both legal and personal situations. The production and distribution of digital photographs have increased exponentially as a result of the quick spread of digital imaging devices like smartphones and digital cameras. Digital images might be just a “picture” for some people, but in the aspect of digital forensics, these images are more than just a picture; they could be evidence that represents facts and criminal ramifications [1]. Besides that, digital images are considered one of the most crucial pieces of evidence in a court case after their legitimacy has been established. Therefore, the analysis of digital images is crucial in the world of digital forensics [2]. In another situation, people with an intention to publish digital images in their social media content might need to check

whether the image they want to share is genuine or fake. In image forensics, image splicing is a common image forgery technique used by perpetrators. The way image splicing works is by copying a part of an image and pasting it onto another image to make it look legit, but it is forged [3]. In other words, it is a type of image forgery that combines two or more images together, producing a single image. In the image splicing research, Error Level Analysis (ELA) is commonly used as a digital forensics' method in helping to detect image tampering. It works by comparing the amount of error that was highlighted to an image during the compression process [4]. The error might consist of the compression levels from different regions in the image. Since the spliced area will have more compression after being edited, it tends to have higher error levels compared to the other areas of the image [5].

Furthermore, common image splicing detection tools only produce a grayscale image with lighter or darker shades to highlight the error in the tampered area, making it hard to distinguish the difference. Similarly, the difference of shades also appears in an original image, which leads to misinterpretation by the users.

On the other hand, some of the tools also do not provide a report analysis to identify whether the image is tampered with or not. Meanwhile, some of the non-web-based tools also require programming knowledge to run the detection application [6]. As a result, users that aren't experts in the image forgery detection field might not know how to read the result. Moreover, as the tools also didn't provide any account for users to track their analysis record, it will take extra time for them to get some advice from the image forensics experts.

Therefore, this project aims to create a web-based image splicing detection tool called SpliceFound, which is convenient for common users and forensics or legal practitioners. This tool can also help content creators verifying the images' authenticity that they will be using as their content.

The paper is structured as follows: Section 2 discusses the literature review related to image splicing forgery tools. Section 3 will explain the methodology used. Section 4 explains the analysis and design of the tool. Section 5 presents implementation and testing while conclusions are drawn in section 6.

2. Literature Review

This section covers the literature review of image forensics, the role of digital forensics in legal processes, and a comparison of existing tools with the proposed tool.

2.1 Digital Forensics

Digital forensics is the process of gathering and analyzing data from computer systems, storage devices, and networks so that it can be used as evidence in court. It is important in legal processes to examine and analyze data to help in legal proceedings. Digital forensics fall under computer forensics. In digital forensics, evidence collection is not done only once but is done in an orderly manner. Digital forensics maintain standards of real-time investigation and evidence collected online. There are two categories of crimes, the first one being when the crime occurs, and the computer used to commit the crime. The second one being when the computer stores evidence and the crime results in harm to the computer system [8]. Investigation of evidence requires Digital Forensic analysis and presentation to be brought in court.

2.2 Image Forensics

A branch of digital forensics is also known as image forensics. It is dedicated to examining digital images in order to confirm their authenticity, identify any tampering, and track down their source. It uses a variety of methods to find signs of tampering, including cloning, retouching, and splicing, which can change the content and originality of an image. In addition to identifying changes, image forensics can identify an image's origin, including the camera or software that was used, and retrieve metadata to learn more about the image's production and modification history. This field is important in detecting fraudulent content, assisting legal and investigative procedures, and guaranteeing the reliability of visual evidence in a variety of fields, such as cybersecurity, content creation, and law enforcement [9]. Tampering detection can be done on image forgeries like copy-move and image splicing.

2.3 Image Forgery Detection

The goal of image forgery detection is to spot alterations in digital images while maintaining their integrity and authenticity. Common techniques of forgeries include image splicing, in which elements from different images are combined to create one seemingly realistic image. Because of the rapid spread of manipulated images online, sophisticated techniques like ELA and machine learning are used to detect these manipulated images. However, ELA is a commonly used technique to detect image manipulation by looking at the compressed image quality. This technique is applied to an image to identify manipulations. On the other hand, image segmentation can be identified by machine learning. Image forgery detection can be useful in forensic investigations [10]. One of the most popular techniques of image manipulation is image splicing. The way image splicing works is by copying a specific region from one image and pasting it onto another image, or what we call a host image. This will

create a seemingly genuine image [11]. To get a spliced image, an image has to go through geometric transformations that result in the image content losing its originality. Although it is hard to distinguish between a genuine image and a spliced image by the naked eye. SpliceFound is concentrating on ELA-based artifact visualization in this project, offering user-friendly features like downloadable reports and account-based analysis histories in addition to distinct visual markers of tampered locations.

2.4 Error Level Analysis

Error Level Analysis (ELA) is a type of image forgery detection technique. It is considered the traditional way of detecting image splicing. It works by detecting inconsistencies in compression levels of an image to identify tampered areas. Higher error levels in areas could indicate image modification such as splicing. Next subsection will discuss three existing image forgery detection tools. Error Level Analysis (ELA) compares an image to its original to find discrepancies after resaving it at a particular compression level, usually 95% [12]. If the compression is consistent throughout all areas of the image, it indicates that the image is probably genuine. However, tampering may be indicated if some regions have higher levels of error. An ELA image that is typically shown in grayscale as a result of this procedure frequently has tampering areas that appear brighter because of increased compression errors. The detection of tampered areas is helped by this visual distinction. Because ELA depends on the unique properties of lossy compression found in JPEG images, it works best when applied to JPEG images.

2.5 Study of Existing Tools

There are three existing tools that are similar to SpliceFound which are FotoForensics, Forensically and ErrorLevelAnalysis.rar. Each of these tools has its advantages that might be referenced in developing the proposed tool.

2.5.1 FotoForensics

FotoForensics is an online tool that allows users to analyze an image to find possible signs of tampering or modification. It uses a few forensic methods, including ELA to find differences in compression levels that can indicate that the image has undergone alterations, or it is tampered. Fig. 1(a) shows FotoForensics homepage. Users can enter image URL or upload an image from their computer. Users must click on “Upload URL” if they entered a link. Users have to click on “Upload File” if they uploaded an image from their computer. A file named “spliced_!” was uploaded from computer.

Surely, there would be some disadvantages to FotoForensic which is it does not colorize the tampered region or state exactly where the tampered region is as shown in Fig. 1(b). On the other hand, FotoForensic stores the history of detection using browser’s cache, not in a server or database which means if the cache is deleted, all the history of detection will be gone.

(a)

(b)

Fig. 1 *FotoForensics homepage(a) and Image result(b)*

2.5.2 Forensically

Forensically is an online tool that is like FotoForensics, it helps user to analyze an image to find possible signs of tampering or modification. It offers many other forensic analysis techniques, one of them being ELA. This tool is suitable for detecting changes by highlighting areas with inconsistent compression levels. Users can upload their image to Forensically to see if their image is spliced or genuine.

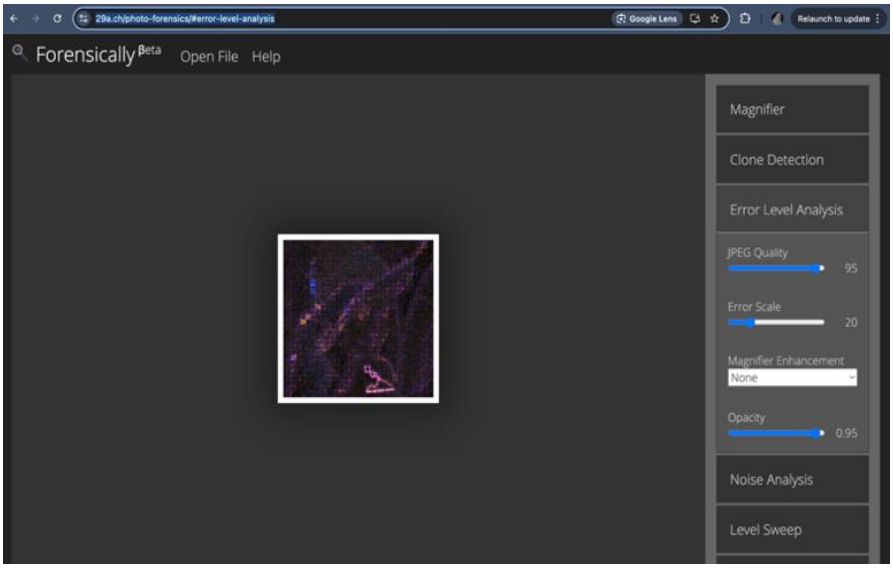


Fig. 2 *Forensically*

Fig. 2 shows Forensically homepage. Users can upload an image by clicking "Open File". Here, the tampered region is highlighted in a bold and bright colour compared to the rest of the image. Users can perform ELA on images as well as other forensic analysis techniques like clone detection, noise analysis, level sweep, string extraction and many more. The disadvantage of Forensically is that it does not provide user login module and image storage and retrieval, therefore, users cannot access history of the image's detection.

2.5.3 ErrorLevelAnalysis.rar

The ErrorLevelAnalysis.rar is a source code for an image splicing detection tool. It is not a web-based tool, it is found on a website and its author's name is Sebastiano Milardo. Fig. 3 shows that the website explained about how to install the source code, the description of what ELA is, how to use the tool and how to debug the install tool. The disadvantage of this tool is that it is not web-based, therefore, user have to download the source code and run it themselves. This might cause confusion to non-technical users as they might not know how to run the code or the need to install a text editor to run the code.

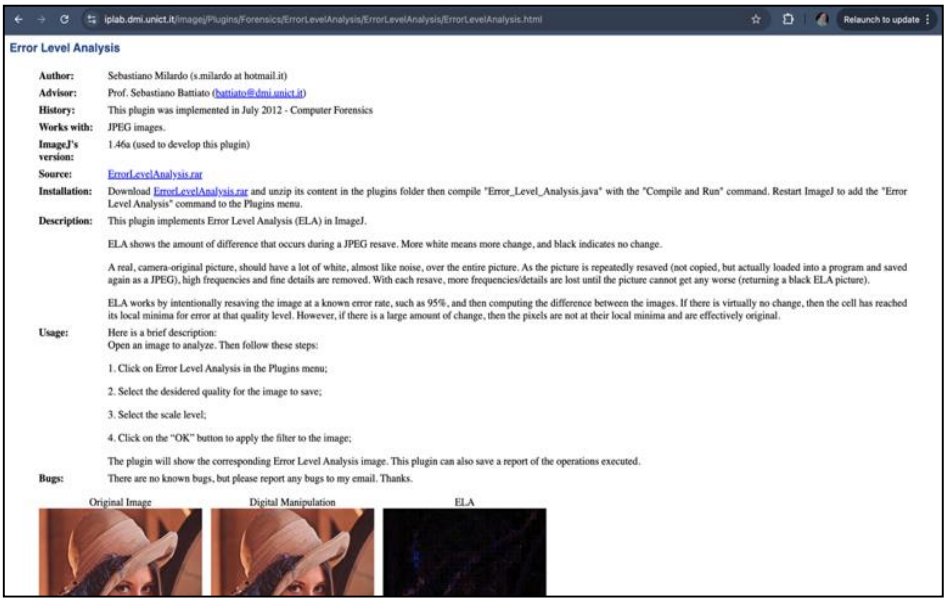


Fig. 3 *ErrorLevelAnalysis.rar*

Three existing tools, FotoForensics, Forensically, and ErrorLevelAnalysis.rar, were examined and compared to SpliceFound in Table 1.

Table 1 *Tool Comparison*

Features \ Tool	FotoForensics	Forensically	ErrorLevelAnalysis.rar	SpliceFound
Type of tool	Web-based	Web-based	Need to download JAVA source code	Web-based
Splicing detection techniques	ELA	ELA, Noise Analysis, Level Sweep	ELA	ELA
Support thresholding and Image masking	X	X	X	√
Report generation	X	X	X	√
Require user account registration and login	X	X	X	√
Image retrieval and history	√	X	X	√

Based on Table 1, FotoForensic, Forensically, and SpliceFound are web-based tools, while ErrorLevelAnalysis.rar requires manual download. All these tools support ELA as its splicing detection technique. However, forensically also supports noise analysis and level sweep techniques. In terms of history access, only SpliceFound requires user account creation. FotoForensic has image retrieval, but the entries are stored in the browser's cache, not in the server or database. Therefore, if the cache of the browser were to be deleted, all the entries would be gone as well. SpliceFound stores and retrieves images from the database. Forensically and ErrorLevelAnalysis.rar do not support image storage and retrieval features.

3. Methodology

This section discusses the methodology used, the task and output of each phase, the tool requirements, the tool workflow, Use Case Diagram, Sequence Diagram, Activity Diagram, Class Diagram and the user interface design. Based on Fig. 4, there are six phases in the Agile Model which includes requirements, design, development, testing, deployment and review. The next subsections will further discuss what activities are done in each phase. The Agile Model is a method that places a strong emphasis on flexibility, iterative development, and active participation from the users to flag the project as success. The Agile Model focuses on short-term planning and flexibility more than traditional methodologies like the waterfall model. This could help in reacting to project requirements and environmental changes. The Agile Model facilitates continuous participation from the users, like receiving their evaluation and feedback with each step taken, leading to on-time delivery and increasing the user's satisfaction. Throughout the whole process, users stay actively involved, which promotes users to feel like they are connected to the tool, and it was made for them [12]. This also enables the tool to have continuous improvement. There are six phases in the Agile phase, as mentioned earlier. Every phase, as illustrated in Table 2, has a specific task and output that needs to be completed during the project development.

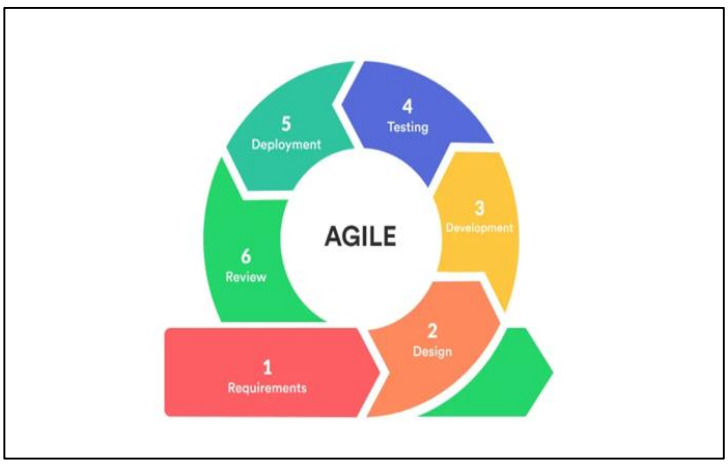


Fig. 4 *Agile Model*

Table 2 *Task and output of each phase*

Phase	Task	Output
Requirement	Identify the problem, identify users to understand project's goals and objectives. A Gantt chart needs to be created	Product proposal, Gantt chart
Design	Functional and non-functional requirements are defined. A user-friendly interface is also designed for the tool. Tool workflow including front-end, back-end, and UML diagrams need to be done.	Functional and Non-Functional Requirement, user interface design of tool, software and hardware requirements, database design and UML diagrams.
Development	Develop the tool and create database	Completed tool with connection to the database
Testing	Conduct testing (user acceptance testing), meet user for testing and test for bugs	Test reports, big fixes and quality assurance to flag the tool as ready for deployment
Deployment	Deploy tool for end user to use on server and monitor tool performance after deployment	User manual for user ease of use
Review	Collect user feedback, monitor any bugs and release patches if needed	Feedback reports from users and bug correction

Based on Table 2, the project development process is divided into six phases, each with specific tasks and outputs. In the requirement phase, problems and target users are identified to define project goals, producing a product proposal and Gantt chart. The design phase focuses on defining functional and non-functional requirements, creating a user-friendly interface, and preparing tool workflows, database design, and UML diagrams. During development, the tool and database are built and integrated. The testing phase involves user acceptance testing, bug fixing, and quality assurance. In the deployment phase, the tool is launched on a server with a user manual for guidance. Finally, the review phase gathers user feedback, monitors performance, and provides updates or patches as needed.

4. Analysis and Design

This section discusses the analysis and design of SpliceFound. This section includes the tool requirements, tool development workflow, Use Case Diagram, Sequence Diagram, Activity Diagram, Class Diagram, image processing flowchart and user interface design.

4.1 Tool Requirements

Functional requirement is the process of determining and recording the precise functions that a tool must have in order to satisfy the needs of its users; it shows what the tool should do, as shown in Table 3. Non-functional requirements tell the behavior of the tool rather than its functionalities. In Table 4**,** the non-functional requirements are discussed based on four groups: performance, security, operational, and usability.

Table 3 *Functional Requirements*

No	Modules	Functionalities
1	User Registration and Login	The tool should allow user to register an account. The tool should allow authorized user to log into their account.
2	Image Upload	The tool should allow user to upload image for analysis
4	Splicing Detection	The tool should be able to detect splicing using ELA, thresholding and image masking
5	Report Generation	The tool should be able to produce a report to state the result of the analysis
6	Results History	The tool should be able to maintain all records of user-uploaded image and its detection result
7	Admin	The tool should allow admin to view users' details like their email, profession, reason of using the tool and upload count.

Table 3 shows the functional requirements. There are seven modules altogether, the first one being User Registration and Login. Where tool needs to allow users to sign up for an account and log into their account. The second one being image upload module where it should allow users to upload images for analysis. Splicing Detection module makes sure that the tool must be able to detect splicing using the specified algorithm. Report Generation module makes sure that the tool can produce report to show analysis result. Moreover, Results History module makes sure that the tool can store user-uploaded image and analysis result. The Admin module makes sure that the tool allows admin to view user details.

Table 4 *Non-Functional Requirements*

No	Requirements	Description
1	Performance	The tool should locate the users on their correct session
2	Operational	The tool should provide a straight-forward onboarding process for new-users. Guide users through registration, image upload and understand the image analysis results.
3	Usability	The tool has a user-friendly interface and is easy to navigate around.

Table 4 shows the non-functional requirements which are performance, operational and usability. SpliceFound is required to locate users on the right session. It should provide a straight-forward process for new users that helps them understand image analysis result. Moreover, it should have a user-friendly interface for user's ease of use.

4.2 Tool Development Workflow

Tool workflow explains the components and procedures of this tool. It outlines the interactions between different modules, features, and interfaces to accomplish the tool's goals. This emphasizes how its components work together to produce the desired results. It outlines the tool's logical structure, including how data moves between modules.

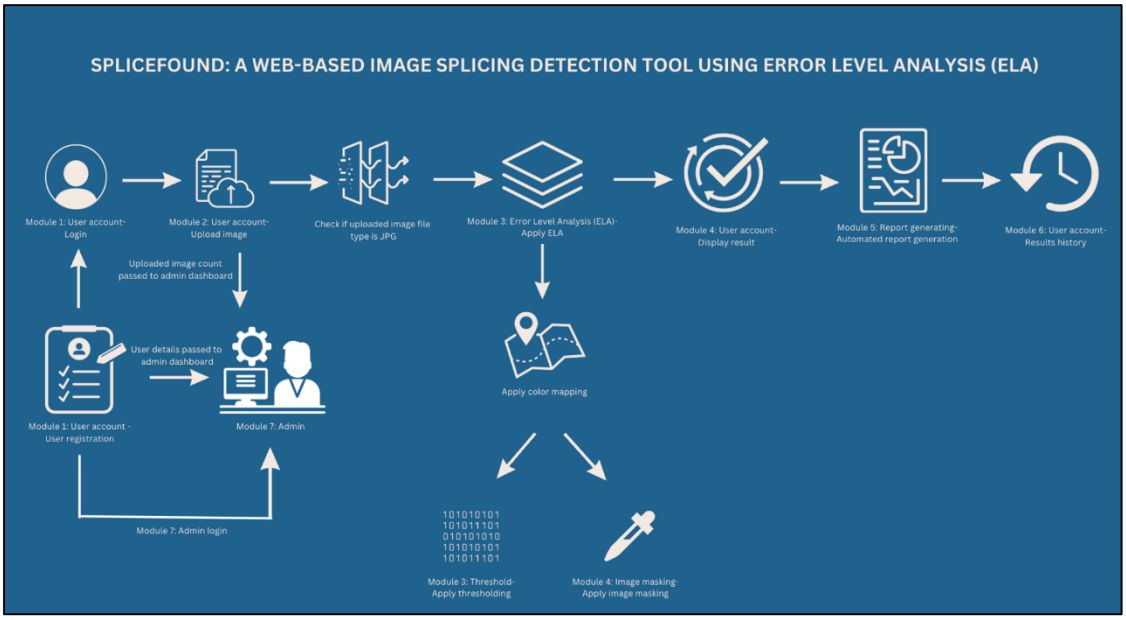


Fig. 5 *Tool workflow*

Fig. 5 shows SpliceFound's workflow. User can create an account by registering. They can log into their account to gain access to SpliceFound. User will upload an image, and it will check the file type to see if it is JPG or not. This part uses file validation. It will restrict file picker to only show JPG files. Then, the image tampering detection process utilizes a combination of Error Level Analysis (ELA) and threshold-based masking to assess the authenticity of an image. To guarantee uniform analysis and performance, the uploaded image is initially resized to a maximum size of 800×800 pixels. To standardize compression artifacts, it is saved in JPEG format with a normalized quality setting of 95%. ELA involves comparing the original image to its recompressed version, with pixel-wise differences computed via ImageChops.difference(). These differences are visually amplified through brightness enhancement and Gaussian blur, making subtle artifacts more noticeable. Then, the ELA-enhanced image is turned into grayscale, and a binary mask is created with a predetermined intensity threshold.

This mask emphasizes the areas with considerable differences, which could suggest tampering. Then, the total number of pixels in the image is compared to the quantity of white pixels (potentially altered areas) in the mask. When the number of highlighted pixels exceeds a set threshold which is 0.02, the image is categorized as "Tampered." In other cases, it is marked as "Genuine." The process of making this decision is based on real images will show minimal compression inconsistencies, while tampered images create unnatural artifacts that ELA can uncover. The analysis result will be displayed by simply stating if the image is tampered or authentic. An automated report is generated which summarizes if the uploaded image is tampered or authentic. The insight of this report will show which area is highlighted and give comparison between uploaded image and image that has undergo ELA. Finally, users can view their past analysis result in results history. This workflow shows how each module and features integrates with each other. On the other hand, admin can login to the dashboard to view user details and image upload count per user.

4.3 Use Case Diagram

A use case diagram is a type of Unified Modelling Language (UML) diagram. It is used to visually show how users (also known as actors). It illustrates what a system is intended to accomplish from the viewpoint of the end-user, rather than detailing how it is accomplished. A use case represents a process like "Login," "Upload Image," or "View Report." Actors typically consist of users that interact with the system. Fig. 6 shows the Use Case Diagram where admin can login and view user's data like email, profession, why are they using SpliceFound and the number of images they have uploaded.

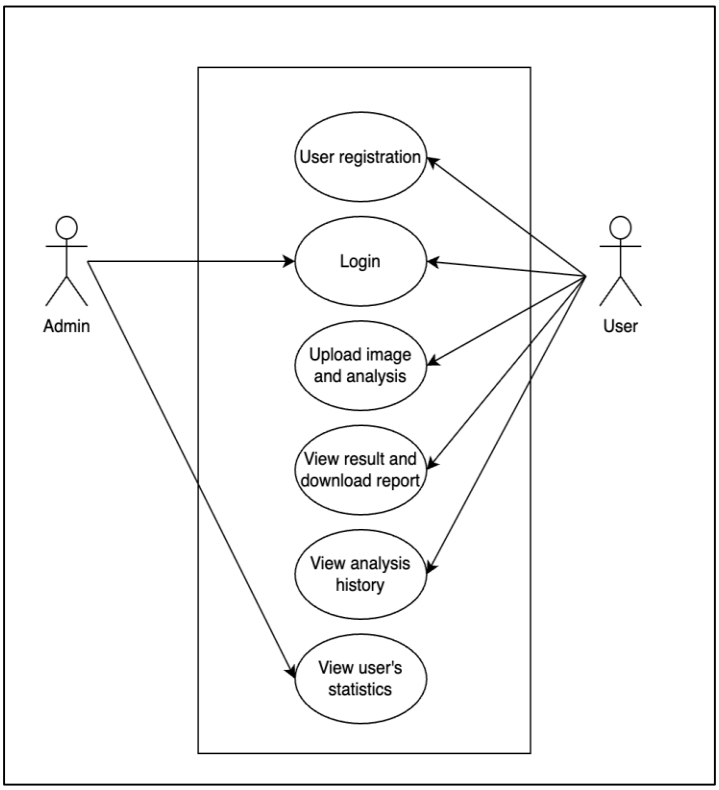


Fig. 6 *Use Case Diagram*

4.4 Sequence Diagram

A sequence diagram is a type of UML diagram that shows how messages move between various objects or components in a system. Primarily, it is employed to simulate the sequential operation and interaction of processes, displaying the chronological sequence of events. A lifeline connects each item or process participant, and arrows signifying messages or function calls are used to illustrate communication between them. Sequence diagrams are therefore particularly helpful for showing the logic of a particular event, like user login, data processing, or system response. The diagram's emphasis on the messages' time sequence aids developers in better understanding system behaviour and creating interactions.

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1015-1036

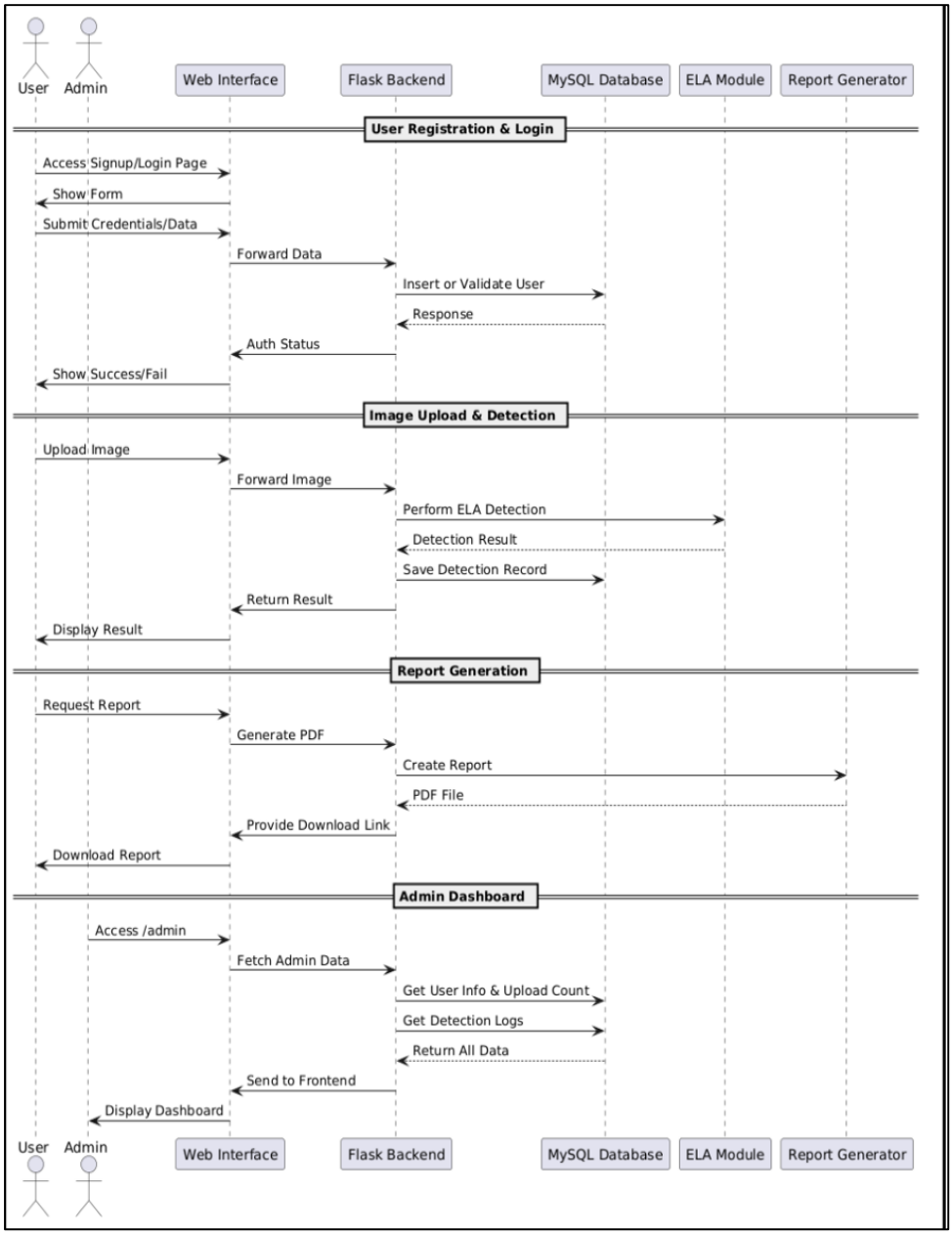


Fig. 7 *Sequence Diagram*

Fig. 7 shows the tool's sequence of how users and administrator interact with the web interface, backend, MySQL database, ELA detection module, and report generator, among other system components. The first step in the process for users is registration, which they provide their details using the signup form.

The backend receives this data, stores it in the MySQL database, and verifies that the registration was successful. After that, users enter their credentials, which are verified against the database in the login page. Users can upload photos for tampering detection after logging in. The backend provides the user with the results after sending the image to the ELA module for analysis and saving the detection results in the database. The backend creates and makes available for downloading a PDF report that summarizes the analysis upon request from the user.

Administrators have access to an admin dashboard. All user information, including user email, profession, reason of use and the number of image uploads per user, are retrieved from the database by the backend for the dashboard. The web interface displays this extensive data, giving administrators the ability to keep an eye on user behavior. Administrators can also ask for a complete list of users together with their responsibilities and other information. Overall, the figure illustrates how the database, backend, web interface, and specialized modules like as ELA and report production work together to support the system's administrative oversight as well as user functions.

4.5 Activity Diagram

Activity Diagram is a type of flowchart created using the Unified Modelling Language (UML) that illustrates how an activity moves from one activity to another. It's called an activity diagram because it outlines what should occur in the modelled system and is used to show the various dynamic characteristics of a system. Fig 8 shows the activity diagram, if the admin is accessing SpliceFound, they will just have to add "/admin" at the end of the link. They can then access user details like user email, profession, reason of using SpliceFound and image upload count. If it is a normal user, they will log into their account and choose whether to upload an image, view result history or logout. If user chooses upload image, that is where the analysis will be performed and results of it will be shown with an option to download report. If user chooses to view result history, user is accessing past analysis result. They can also download reports here. If users choose to log out, they will be logged out from their account and directed to the login page.

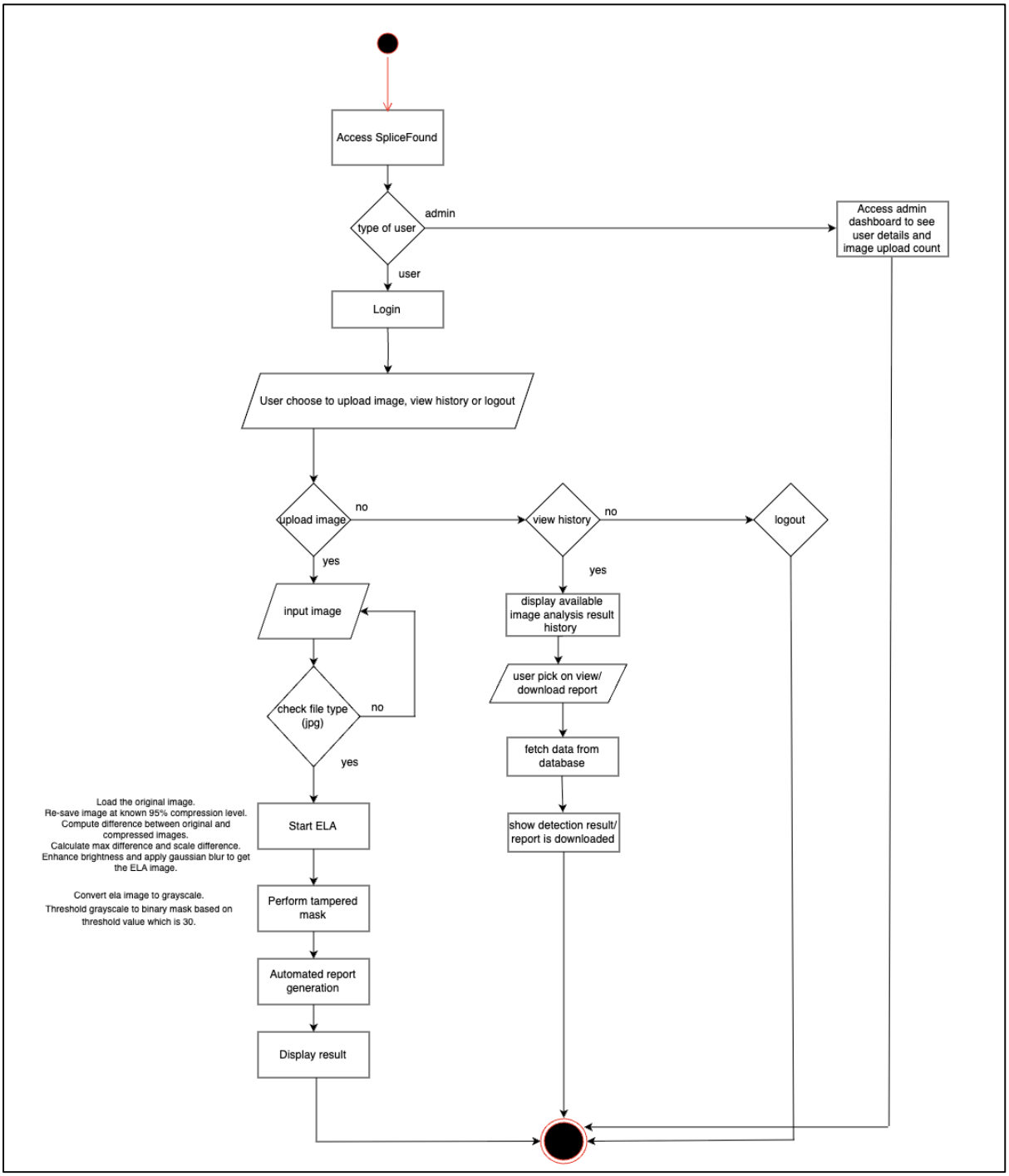


Fig. 8 *Activity Diagram*

4.6 Class Diagram

A class diagram is a type of visual representation that illustrates a tool's structure. As a component of the Unified Modeling Language (UML), it shows the classes in a program as well as their methods (functions) and attributes (data). It also illustrates the connections between various classes, including which classes inherit from or depend on one another. This aids developers in comprehending the connections between various program components and the data flow within the system. They serve as a blueprint, providing an orderly and transparent perspective of the tool's development.

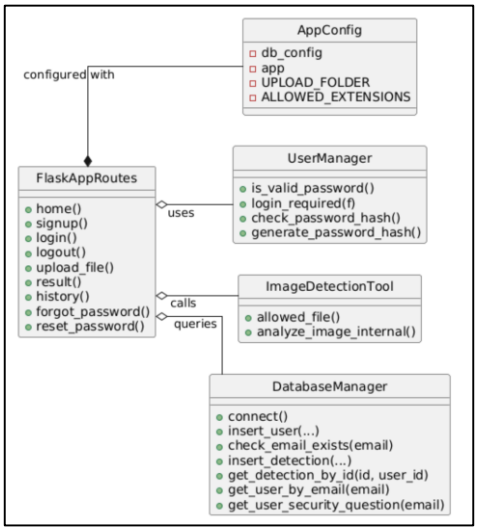


Fig. 9 *Class Diagram*

Fig. 9 shows the Class Diagram for SpliceFound. SpliceFound is divided into five primary classes in this diagram. FlaskAppRoutes are configured by AppConfig. All user interaction route functions are defined by FlaskAppRoutes. FlaskAppRoutes uses UserManager for user authentication. FlaskAppRoutes calls ImageDetectionTool to analyze uploaded image. FlaskAppRoutes queries DatabaseManager to acess or store data.

4.7 Image Processing Flowchart

The image processing flowchart (Fig. 10) shows how the image that has been uploaded by user is being processed from a JPG image to ELA image then being masked using image masking and thresholding as well as generating a report for a more detailed analysis in SpliceFound. When a user uploads an image, the tool saves it and rescales it to a maximum size of 800x800 pixels to standardize processing. Then, the image is normalized by saving it as a JPEG with a quality setting of 95%. After that, the process of Error Level Analysis (ELA) is carried out by re-compressing the image at the same quality level and determining the pixel-wise difference between the original and compressed versions of the image. The equation given demonstrates how ELA works. In the expression, 'I' denotes a JPEG image. As a single instance of ELA calculation, let us consider An representing a JPEG that has been resaved n times using a 75% quality setting, and Bn representing a JPEG that has been recompressed n times with a 95% quality setting. Therefore, the mentioned ELA calculation can be referred to as "using ELA of 95%". Other than that, the task is to investigate what the error value will be if the JPEG is recompressed using a compression quality of 95%.

As the JPEG is resaved 1, 2, ..., n times, the extent of compression error ELA1, 2..., n will decrease. Due to resaving times, each 8 × 8 JPEG block will gradually approach its local minima and darken (Azhan, et al., 2022). This difference is enhanced by brightness adjustment and applying a slight blur to highlight potential tampered areas. To produce a binary tamper mask that indicates pixels probably subject to modification, the enhanced ELA image is transformed into grayscale and thresholded. The original image is then overlaid with this mask, which highlights suspicious areas in white. To estimate the probability of tampering, the system calculates the ratio of tampered pixels to the total number of pixels in the image. When this ratio goes above a set limit (such as 2%), the image is marked as "Tampered" if not, it is marked as "Genuine." Finally, the ELA image, tamper mask, and highlighted image are saved for reference, while the tampering results are returned for storage and display. The report is auto generated, ready to be downloaded if a user needs it.

As for the metadata extraction process, by using Image.open(P), where P is the file path, to load the image is the first step in the metadata extraction process. After retrieving and storing the file format, os.path.getsize(P) is used to determine the file size in bytes. The width and height of the image are accessed to determine its resolution. I._getexif() is then used to retrieve the EXIF (Exchangeable Image File Format) metadata. The date the picture was

shot (DateTimeOriginal or fallback DateTime) and the camera model (Model) are obtained and saved if EXIF data is present. Default values like "Unspecified" for the camera model and "N/A" for the date are used in the absence of EXIF data. This metadata aids in supplying crucial details regarding the qualities and place of origin of the image.

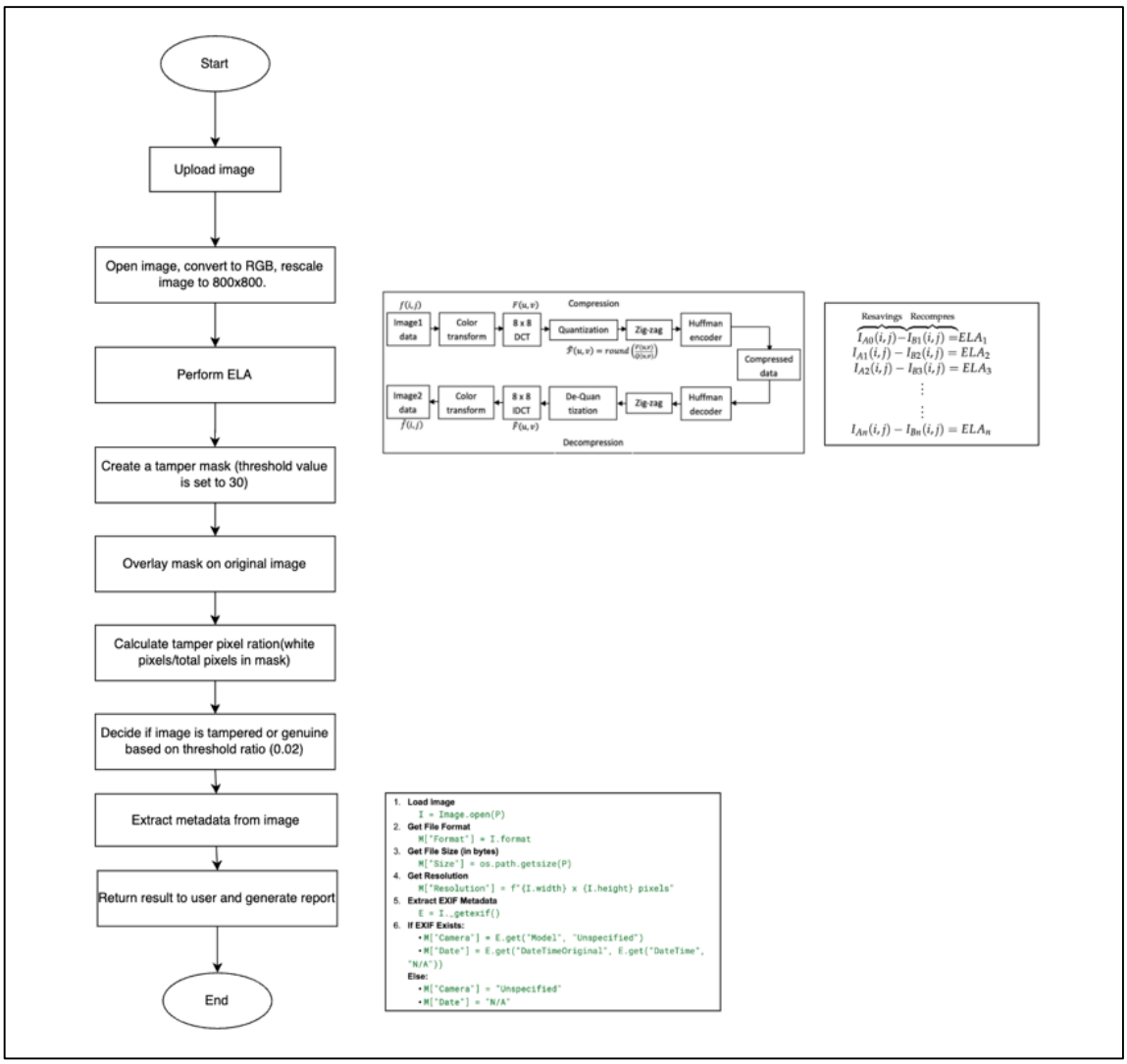


Fig. 10 *Image Processing Flowchart*

4.8 Database Design

A comprehensive list of the data elements utilized by the tool is provided by the Data Dictionary. It offers explanations of every component and acts as a reference for the database's relationships and data types. Table 5 and 6 shows the detailed data dictionary which includes the table names, data type, size, key and its description.

Table 5 *users table*

Attribute	Data Type	Size	Key	Description
id	INT	-	Primary Key	Unique identifier for each user
email	VARCHAR	255	Unique	User's email address
password	VARCHAR	255	-	Encrypted user password
profession	VARCHAR	255	-	User's profession
reason	TEXT	-	-	Reason for using the system
security_question	VARCHAR	255	-	Security question for account recovery
security_answer_hash	VARCHAR	255	-	Hashed answer to the security question

Table 6 *detections table*

Attribute	Data Type	Size	Key	Description
id	INT	-	Primary Key	Unique identifier for each detection record
user_id	INT	-	Foreign Key	References the associated user in users(id)
original_image	VARCHAR	255	-	Path to the original uploaded image
ela_image	VARCHAR	255	-	Path to the ELA-generated image
mask_image	VARCHAR	255	-	Path to the binary mask image (if generated)
highlighted_image	VARCHAR	255	-	Path to the image with tampered area highlighted
result	VARCHAR	50	-	Analysis result (e.g., Tampered, Genuine)
tamper_ratio	FLOAT	-	-	Ratio of tampered pixels to total pixels
timestamp	TIMESTAMP	-	-	Timestamp when analysis was performed
image_path	VARCHAR	255	-	Full path to save original image
ela_path	VARCHAR	255	-	Full path to save ELA image
highlighted path	VARCHAR	255	-	Full path to saved highlighted result image

5. Implementation and Testing

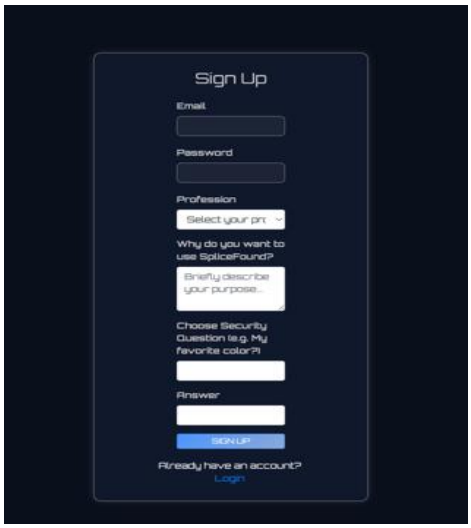
Implementation is the process of applying the design that was made to develop a whole tool while testing is the process of finding out whether the developed tool is working as intended.

5.1 Implementation

This section covers the implementation of modules that were mentioned in Table 3 which are user account registration, login, image upload, splicing detection, display result, report generation, result history and admin.

5.1.1 Sign Up

Fig. 11(a) shows SpliceFound's sign-up page. Users must enter their email and password for login purposes. Profession and reasons of using SpliceFound for data collection purposes. Lastly, security questions and answer for forgot password purposes. After they are done with filling in the form, they can click on "Sign up". If users already have an account, they can click on "Login" to be directed to the login page. Fig. 11(b) shows the python source code for signup page. This code is where the logic for signup page is being put at. It posts all the data given by the user to the database. It hashes user's password and the security answer. It also has a function to check if an email already exists to make sure user account is not created more than once. It will flash a message if the signup is successful. All the input boxes are required to be filled by the user. It is linked to the signup HTML file for displaying the page. Fig. 11(c) shows the JavaScript code for the signup page where it checks if user's password follows the strong password guidelines. A password must be more than 8 characters, includes uppercase and lowercase letters with numbers and symbols.



(a)

```python
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        profession = request.form['profession']
        reason = request.form['reason']
        security_question = request.form['security_question']
        security_answer = request.form['security_answer']

        # Hash password and security answer
        password_hash = generate_password_hash(password)
        security_answer_hash = generate_password_hash(security_answer.lower())  # normalize to lowercase

        # Check if email exists
        conn = mysql.connector.connect(**db_config)
        cursor = conn.cursor()
        cursor.execute("SELECT id FROM users WHERE email = %s", (email,))
        if cursor.fetchone():
            flash('Email already registered.')
            cursor.close()
            conn.close()
            return redirect(url_for('signup'))

        # Insert user
        cursor.execute("""
            INSERT INTO users (email, password, profession, reason, security_question, security_answer_hash)
            VALUES (%s, %s, %s, %s, %s, %s)
        """, (email, password_hash, profession, reason, security_question, security_answer_hash))
        conn.commit()
        cursor.close()
        conn.close()

        flash('Sign up successful! Please log in.')
        return redirect(url_for('login'))

    return render_template('signup.html')
```

(b)

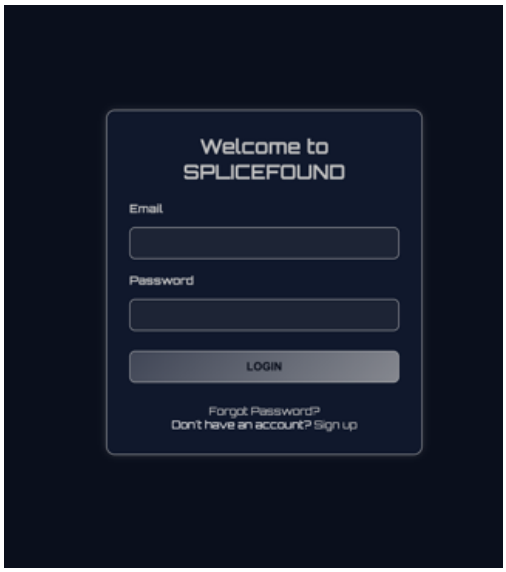
```
<script>
    function validateForm() {
        const password = document.querySelector('input[name="password"]').value;
        const pattern = /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{8,}$/;

        if (!pattern.test(password)) {
            alert("Password must be at least 8 characters long, include uppercase and lowercase letters, a number, and a special character.");
            return false;
        }
        return true;
    }
</script>
```

(c)

Fig. 11 *Signup page(a), python code(b) and JavaScript code(c)*

5.1.2 Login

Fig. 12(a) shows SpliceFound's login page. Users must enter their registered email and password. If they forgot their password, they could click on the forgot password button to be directed to the forgot password page. If a user does not have an account yet, they can sign up by clicking the "sign up" sign. Fig. 12(b) shows the python source code for login page. This code is where the logic for login page is being put at. It checks if the entered email and password match the one stored in the database. If the credentials are invalid, a flash message or error message will be shown saying "Invalid login credentials. Please try again". It is linked to the HTML file for displaying the page.



(a)

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']
        conn = mysql.connector.connect(**db_config)
        cur = conn.cursor(dictionary=True)
        cur.execute("SELECT * FROM users WHERE email = %s", (email,))
        user = cur.fetchone()
        cur.close()
        conn.close()
        if user and check_password_hash(user['password'], password):
            session['user_id'] = user['id']
            session['email'] = user['email']
            return redirect(url_for('home'))
        else:
            flash('Invalid login credentials. Please try again.', 'error')
    return render_template('login.html')

@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))
```

(b)

Fig. 12 *Login Page(a) and python source code(b)*

5.1.3 Image Upload

Fig. 13(a) shows SpliceFound's homepage. Here, users can upload their image for tampering detection by clicking on "Choose file" and uploading image from their computer. After that, users can click on "Upload and Analyze" button to upload the image. There is also a disclaimer that states high resolution images will result in better accuracy. Fig. 13(b) shows the python source code for SpliceFound's homepage. Here, it checks if the uploaded file is jpg or not. If it is not, a flash message will be displayed. All the detection results are stored in the database for result extraction and report generation. This code is where the logic for homepage is being put at. This is linked to the home HTML page.

(a)

```python
@app.route('/upload', methods=['POST'])
@login_required
def upload_file():
    file = request.files.get('image')
    if not file or file.filename == '':
        flash('No file selected')
        return redirect(url_for('home'))

    if allowed_file(file.filename):
        filename = datetime.now().strftime('%Y%m%d_%H%M%S_') + secure_filename(file.filename)
        path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(path)

        with open(path, 'rb') as f:
            img_data = f.read()

        result = analyze_image_internal(img_data, filename)

        conn = mysql.connector.connect(**db_config)
        cur = conn.cursor()
        cur.execute(
            """
            INSERT INTO detections
            (user_id, original_image, ela_image, mask_image, highlighted_image, result, tamper_ratio,
            image_path, ela_path, highlighted_path)
            VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
            """,
        (
            session['user_id'],
            filename,
            f"ela_{filename}",
            f"mask_{filename}",
            f"highlighted_{filename}",
            result['result'],
            float(result['tamper_ratio']),
            filename,                  # image_path
            f"ela_{filename}",         # ela_path
            f"highlighted_{filename}"  # highlighted_path
        )
    )

        conn.commit()
        detection_id = cur.lastrowid
        cur.close()
        conn.close()

        return redirect(url_for('result', detection_id=detection_id))

    flash('Invalid file type')
    return redirect(url_for('home'))
```

(b)

Fig. 13 *Homepage(a) and python source code(b)*

5.1.4 Splicing Detection

Fig. 14 shows the splicing detection python source code where images are rescaled to the maximum size to make all uploaded images uniformed. The perform_ela function is where the ELA algorithm is applied. Image is rescaled to the quality of 95% and all images that have been processed is being stored in a file to be used later during the display result part.

```python
# === Image Analysis Functions ===
def rescale_image(image, max_size=800):
    """
    Rescale image maintaining aspect ratio, so the largest dimension is max_size.
    """
    original_size = image.size
    image.thumbnail((max_size, max_size), Image.Resampling.LANCZOS)
    print(f"[INFO] Rescaled image from {original_size} to {image.size}")
    return image

def perform_ela(image_path, quality=95):
    original = Image.open(image_path).convert("RGB")
    # Rescale input image to max 800x800 before ELA
    max_size = 800
    original.thumbnail((max_size, max_size), Image.Resampling.LANCZOS)

    buffer = io.BytesIO()
    original.save(buffer, "JPEG", quality=quality)  # Normalize JPEG quality to 95%
    buffer.seek(0)
    compressed = Image.open(buffer)

    ela_image = ImageChops.difference(original, compressed)

    extreme = ela_image.getextrema()
    max_diff = max([ex[1] for ex in extreme])
    scale = 60.0 if max_diff == 0 else min(255.0 / max_diff, 60.0)

    boosted_scale = scale * 1.8
    ela_enhanced_color = ImageEnhance.Brightness(ela_image).enhance(boosted_scale).filter(ImageFilter.GaussianBlur(radius=2))

    return original, ela_enhanced_color
```

Fig. 14 *Splicing detection python source code*

5.1.5 Display Result

Fig. 15 shows SpliceFound's results page where the result of analysis will be shown as either "Genuine" or "Tampered". In this case, Tampered. There are four images, the first one is the original image that the user uploaded. The ELA image is the second one. The third image is the black and white mask where only white traces can be seen with a black background. The fourth image is the result image where the white traces are applied to the original image so users can see the tampered region better in the original image. Lastly, users can download their automated report by clicking on the "download report" button. To view past analysis history, users can click on "History", to go back to homepage, users must click on "home". Fig. 16 shows the python source code for displaying result page. This is where all the logic for display result pages are being put at. This code is linked to the HTML page. A detection id that was generated during the analysis is needed to make sure it displays the correct detection result.

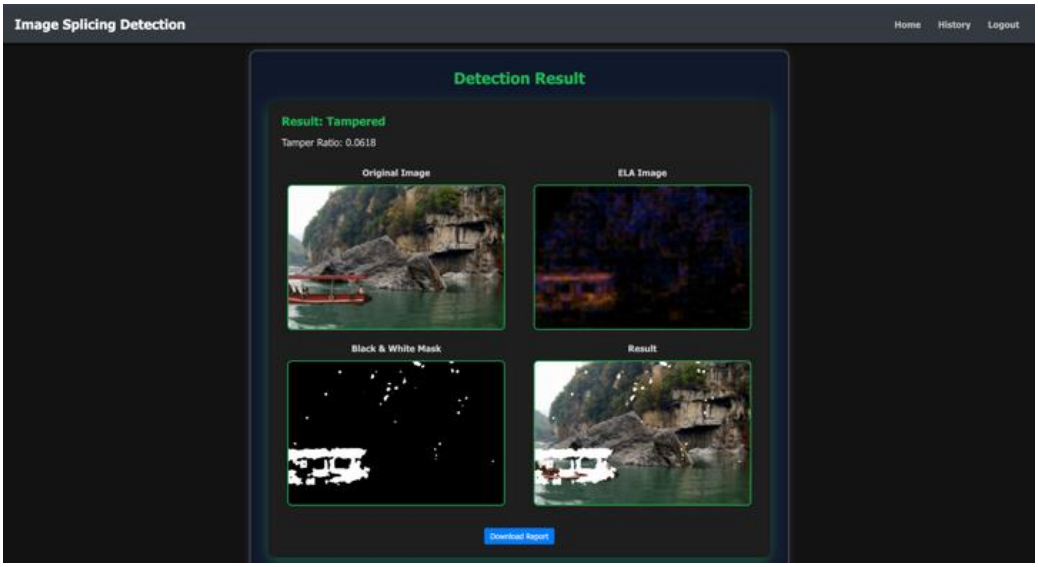


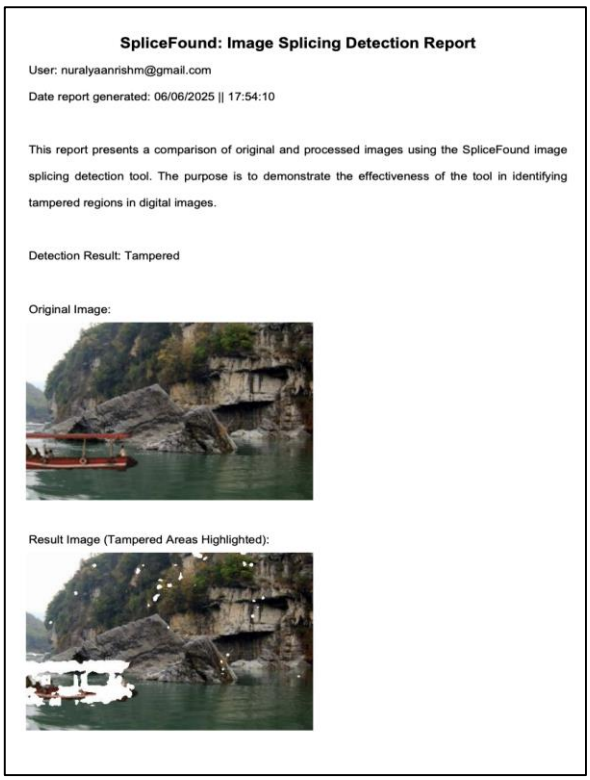
Fig. 15 *Display result page*

```python
@app.route('/result/<int:detection_id>')
@login_required
def result(detection_id):
    conn = mysql.connector.connect(**db_config)
    cur = conn.cursor(dictionary=True)
    cur.execute("SELECT * FROM detections WHERE id = %s AND user_id = %s", (detection_id, session['user_id']))
    data = cur.fetchone()
    cur.close()
    conn.close()

    if data:
        filename = data['image_path']
        data['original'] = filename
        data['ela'] = f"ela_{filename}"
        data['mask'] = f"mask_{filename}"
        data['highlighted'] = f"highlighted_{filename}"
        return render_template('result.html', detection=data)

    flash('Detection not found.')
    return redirect(url_for('home'))
```

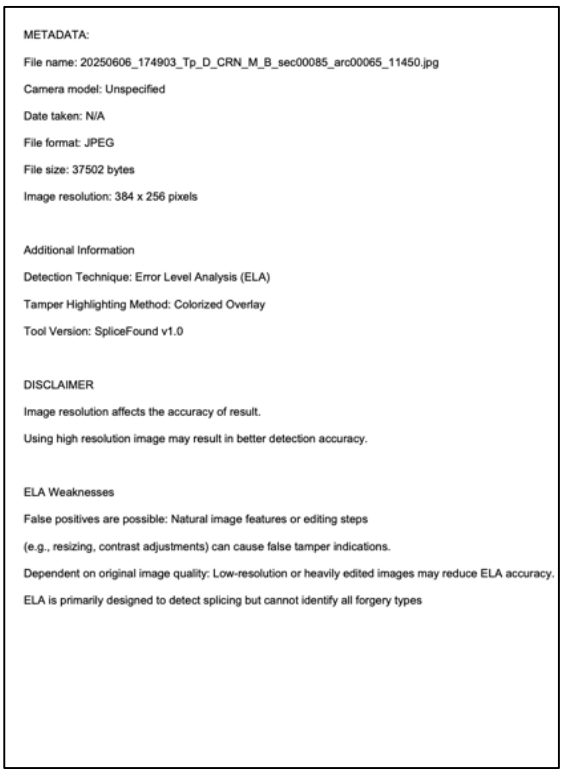
Fig. 16 *Display result page python source code*

5.1.6 Report Generation

Fig. 17(a) shows the report of analysis that is generated by SpliceFound. User email, timestamp, a short paragraph of what the report does, image detection result, original image and result image is shown in the first page. Fig. 17(b) shows the second page of SpliceFound's analysis report. It shows the images of EXIF METADATA, information of SpliceFound, disclaimer and ELA weaknesses. Fig. 17(c) shows the python source code for analysis report page. This part of code is the logic behind the report's first page. Fig. 17(d) shows SpliceFound's EXIF METADATA extraction logic. Here, camera model, date of image taken, file format, file size and image resolution is extracted.



(a)



(b)

```python
@app.route('/generate_report/<int:detection_id>')
@login_required
def generate_report(detection_id):
    conn = mysql.connector.connect(**db_config)
    cur = conn.cursor(dictionary=True)
    cur.execute('''
        SELECT d.*, u.email FROM detections d
        JOIN users u ON d.user_id = u.id
        WHERE d.id = %s AND u.id = %s
    ''', (detection_id, session['user_id']))
    d = cur.fetchone()
    cur.close()
    conn.close()

    if not d:
        flash("Report not found")
        return redirect(url_for('history'))

    # Image Paths
    original_image_path = os.path.join(app.config['UPLOAD_FOLDER'], d['image_path'])
    ela_image_path = os.path.join(app.config['UPLOAD_FOLDER'], f"ela_{d['image_path']}")
    highlighted_image_path = os.path.join(app.config['UPLOAD_FOLDER'], f"highlighted_{d['image_path']}")

    # Detection Result (e.g., Tampered or Not)
    detection_result = "Tampered" if d.get('result') == 'Tampered' else "Genuine"
```

(c)

```python
# --- Insert EXIF extraction code here ---
image = Image.open(original_image_path)

camera_model = "Unspecified"
date_taken = "N/A"
file_format = image.format

exif = {}  # Initialize

exif_data = image._getexif()
if exif_data:
    exif = {
        ExifTags.TAGS.get(tag): value
        for tag, value in exif_data.items()
        if tag in ExifTags.TAGS
    }
    camera_model = exif.get('Model', camera_model)
    date_taken = exif.get('DateTimeOriginal', exif.get('DateTime', date_taken))

file_size = os.path.getsize(original_image_path)


# Open the image file to get its resolution
image_path = os.path.join(app.config['UPLOAD_FOLDER'], d['image_path'])
with Image.open(image_path) as img:
    image_width, image_height = img.size
    image_resolution = f"{image_width} x {image_height} pixels"

# --- End EXIF extraction ---
```

(d)

Fig. 17 *Report first page(a), report second page(b), report python code(c), metadata extraction code(d)*

5.1.7 Result History

Fig. 18 (a) shows SpliceFound's result history page where users can view their past analysis and download report (if they have not been downloaded in the analysis page). Here, users can see their uploaded image filename, timestamp of detection, result of detection and actions whether to just view result or download the report. To go back to the homepage, users can click on "Home". Fig. 18 (b) shows the python source code for result history page in SpliceFound. This is the logic behind the history page. This code is linked to the history HTML page.

(a)

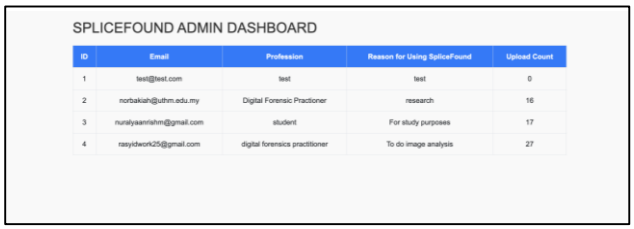
```python
@app.route('/history')
@login_required
def history():
    conn = mysql.connector.connect(**db_config)
    cur = conn.cursor(dictionary=True)
    cur.execute("SELECT * FROM detections WHERE user_id = %s ORDER BY timestamp DESC", (session['user_id'],))
    rows = cur.fetchall()
    cur.close()
    conn.close()
    return render_template('history.html', detections=rows)
```

(b)

Fig. 18 *Result history page(a) and result history python source code(b)*

5.1.8 Admin page

Fig. 19 (a) shows SpliceFound's admin page. Here, admin can see users' email, their profession, the reason of them using SpliceFound and upload images count. Admin page is accessible when adding "/admin" to SpliceFound's link. Fig. 19 (b) shows the python source code for SpliceFound's admin page. This is the logic behind the page. This code is linked to the admin HTML page.




```python
@app.route('/admin')
def admin():
    conn = mysql.connector.connect(**db_config)
    cur = conn.cursor()

    # Updated query: get email, profession, reason, and upload count
    cur.execute("""
        SELECT users.id, users.email, users.profession, users.reason, COUNT(detections.id) as upload_count
        FROM users
        LEFT JOIN detections ON users.id = detections.user_id
        GROUP BY users.id, users.email, users.profession, users.reason
    """)

    users = cur.fetchall()
    conn.close()
    return render_template('admin.html', users=users)
```

(a) (b)

Fig. 19 *Admin page(a) and admin page python source code(b)*

Applied Information Technology and Computer Science Vol. 7 No. 1 (2026) p. 1015-1036

5.2 Testing

The testing for SpliceFound involved User Acceptance Testing (UAT). The UAT was done with the help of SpliceFound's target users. The aim of this testing is to see if SpliceFound qualifies all the requirements needed by the users and to ensure that SpliceFound is working without any issues and is user-friendly. The test was done via Google Form where users can access SpliceFound, upload images using the given dataset and answer the test questions once they are done exploring SpliceFound. The questions and users for this UAT are as seen in the Appendix. The users recruited for this testing were a Digital Forensic Practitioner, a Legal Officer, a Content Creator and a Student. The results are discussed in Table 5 and Table 6.

Table 5 *User Acceptance Testing Result*

Category	Feedback Summary	Category
Account Creation/Login	All users reported successful account registration and login	Account Creation/Login
Image Upload	Smooth and clearly visible upload functionality	Image Upload
Tampering Detection Accuracy	Tools correctly identified and highlighted tampered regions	Tampering Detection Accuracy

Table 6 *Areas for improvement and user suggestion*

Area	User Comments & Suggestions	Area
Report Content	Add file format variety, tampering ratio threshold, image source, capture time, location, camera model.	Report Content
Interface Issues	Login form is small and requires a zoom. Fancy font is hard to read	Interface Issues
Visual Design	- Add color variety - Make buttons more noticeable - Use popup or hover effects	Visual Design

Table 5 shows the User Acceptance Testing result where all the categories and modules are satisfied. Table 6 shows the summary of comments from users for areas of improvements where users commented most on the visuals of SpliceFound that needed more improvements apart from enhancing metadata extraction.

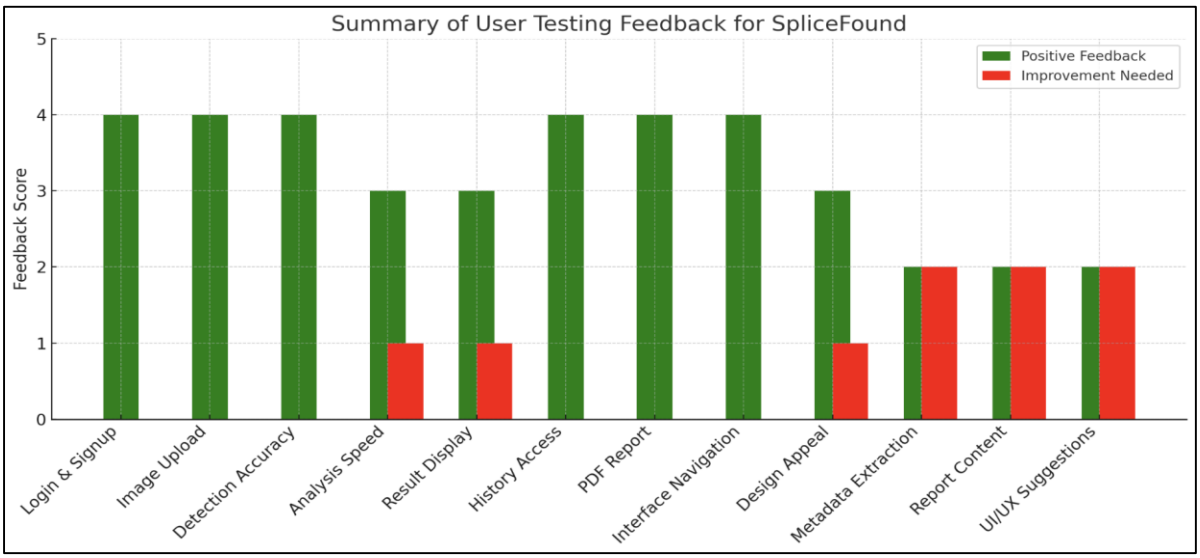


Fig. 20 *Summary of User Acceptance Testing results*

Fig. 20 shows that overall, the tool was rated well in several important functional areas. Users are satisfied with the ability to generate PDF reports, access to analysis history, tampering detection accuracy, image upload experience, and login and signup process, suggesting a solid basis in basic functions. Additionally, the interface's accessible buttons and clear directions made it straightforward to use. Some users reported some delays and minor trouble interpreting marked tampered regions, despite the fact that the analysis speed and result display

were typically evaluated moderately high. Although the interface's overall aesthetic appeal was deemed good, recommendations were given to make it more vibrant and captivating. Users specifically requested better metadata extraction, including information about the camera model, capture time, and image source. Along with fixing certain User Interface issues including small fonts and button visibility, they suggested improving the report's organization and clarity. These observations suggest that overall, the user experience is good, with room for improvement in terms of visual design and the tool's forensic reporting features.

6. Conclusion

In conclusion, SpliceFound's development illustrates a basic method for digital image forensics by the application of Error Level Analysis (ELA). Users can use the system to identify possible tampering in JPEG images, as it highlights inconsistent compression levels that often suggest splicing. SpliceFound provides a structured and user-friendly interface for those looking to verify the authenticity of digital images, with features like user registration, login, image upload, display of detection results, report generation, and a result history module. Despite its limitations like dependence on JPEG format, the need for manual interpretation of ELA results, and lack of automated classification, SpliceFound proves to be a valuable educational and research tool, especially for students, researchers, content creators and professionals in digital forensics and the legal field. Future enhancements could include support for a wider range of image formats and also videos, the use of modules for improved detection, and advanced reporting capabilities. In summary, SpliceFound establishes a foundation for additional investigation into identifying image tampering and makes a significant contribution to the increasing need for verification of digital content's authenticity.

Acknowledgement

The authors express their gratitude to Universiti Tun Hussien Onn Malaysia's Faculty of Computer Science and Information Technology for its support and assistance.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *N. A. Nori Sham, N. B. Abd Warif;* ***data collection*** *N. A. Nori Sham, N. B. Abd Warif;* ***analysis and interpretation of results:*** *N. A. Nori Sham, N. B. Abd Warif;* ***draft manuscript preparation:*** *N. A. Nori Sham, N. B. Abd Warif. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] D. P. Gangwar and A. Pathania, "Authentication of digital image using EXIF metadata and decoding properties," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 3, no. 8, pp. 335–341, 2018. doi: 10.32628/cseit183815.

[2] N. Mesic and J. Hamzabegovic, "Digital image as evidence in case law and software for forensics of digital image," *ResearchGate*, 2022. [Online]. Available: https://www.researchgate.net/publication/359732506_Digital_image_an_evidence_in_case_law_and_software_for_forensics_of_digital_image.

[3] K. B. Meena and V. Tyagi, "Image forgery detection: Survey and future directions," in *Springer eBooks*, 2019, pp. 163–194. doi: 10.1007/978-981-13-6351-1_14.

[4] K. B. Meena and V. Tyagi, "Image forgery detection: Survey and future directions," *Springer eBooks*, pp. 163–194, 2019. doi: 10.1007/978-981-13-6351-1_14.

[5] "Image forgery detection using error level analysis and deep learning," *ResearchGate*, n.d. [Online]. Available: https://www.researchgate.net/publication/332561655_Image_forgery_detection_using_error_level_analysis_and_deep_learning.

[6] S. Milardo, "Error Level Analysis," 2024. [Online]. Available: https://iplab.dmi.unict.it/imagej/Plugins/Forensics/ErrorLevelAnalysis/ErrorLevelAnalysis/ErrorLevelAnalysis.html.

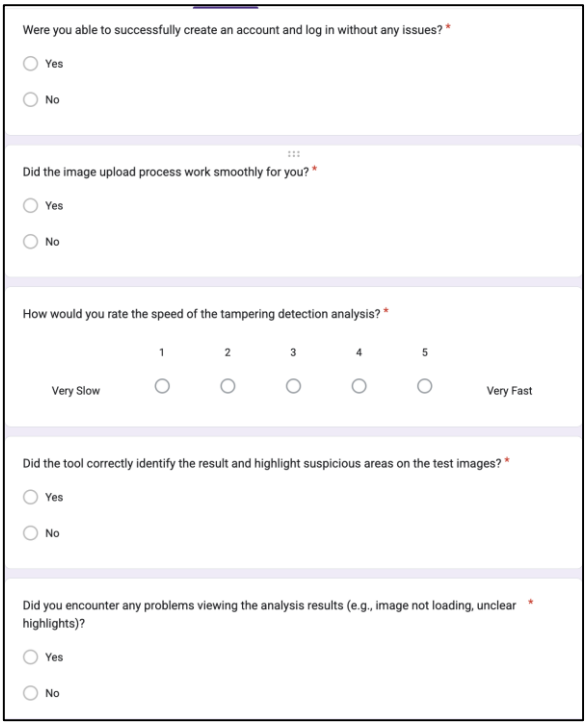
[7] S. Sachdeva, B. L. Raina, and A. Sharma, "Analysis of digital forensic tools," *Journal of Computational and Theoretical Nanoscience*, vol. 17, no. 6, pp. 2459–2467, 2020. doi: 10.1166/jctn.2020.8916.

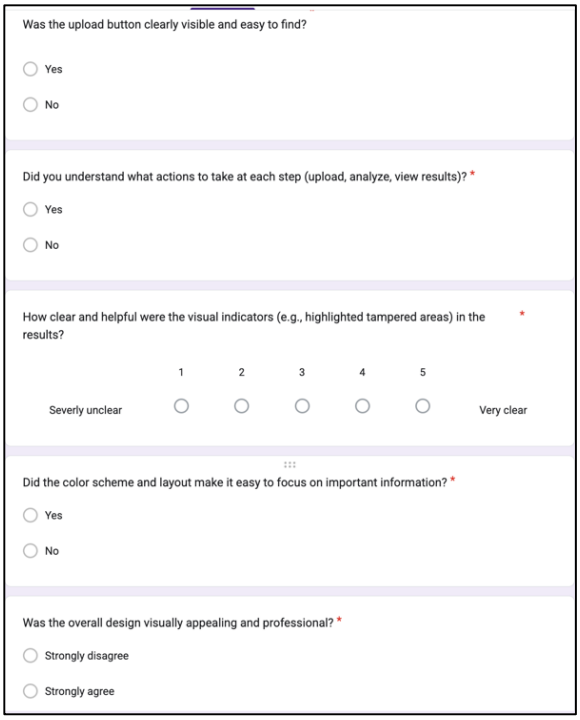
[8] A. Piva, "An overview on image forensics," *ISRN Signal Processing*, vol. 2013, pp. 1–22, 2013. doi: 10.1155/2013/496701.

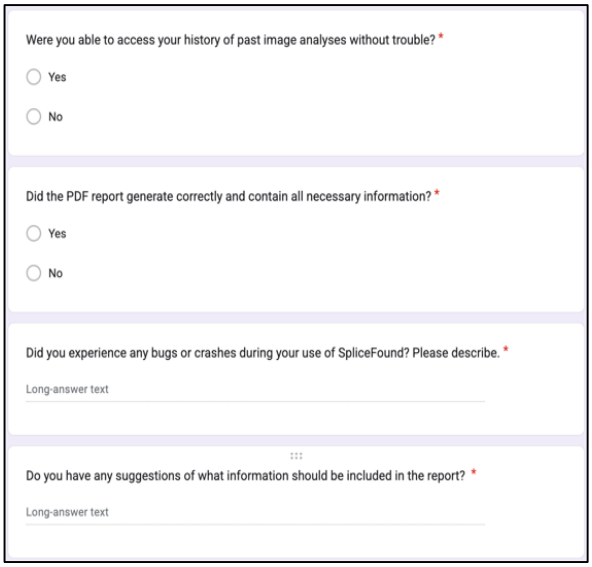
[9] S. Patekar, S. Khan, D. Bhusare, M. Bhujbal, and G. Hegde, "Image forgery detection," 2023. doi: 10.13140/RG.2.2.32571.59680.

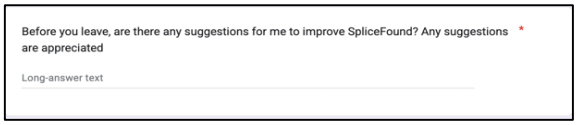
[10] D. Zhang, N. Jiang, F. Li, J. Chen, X. Liao, G. Yang, and X. Ding, "Multi-scale noise-guided progressive network for image splicing detection and localization," *Expert Systems with Applications*, vol. 257, p. 124975, 2024. doi: 10.1016/j.eswa.2024.124975.
[11] K. C. Peitl and C. M. de Oliveira Baptista, "Agile methodology: Benefits and barriers on its initial application," *SAE Technical Paper Series*, 2017. doi: 10.4271/2017-36-0145.
[12] A. A. G. Sudiatmika, I. N. A. Prahara, and I. P. G. Suyadnya, "Image forgery detection using error level analysis and deep learning," *2019 International Conference on Cybernetics and Intelligent System (ICORIS)*, IEEE, 2019. doi: 10.1109/ICORIS.2019.8874887.

Appendix










Name: *
Iffah Radzi
Email: *
iffah73@gmail.com
Your profession *
Content Creator
Name: *
Rasyid Tahir
Email: *
rasyidredha@cybersecurity.my
Your profession *
Digital forensic practitioner
Name: *
NAFISAH BINTI HAMZAH
Email: *
nafisah.hamzah@gmail.com
Your profession *
Legal officer
Name: *
NURUL QISMINA BINTI ROSDI
Email: *
mynarosdi@gmail.com
Your profession *
content creator