

# National College of Ireland

BSHCYB4

Cybersecurity

2023/2024

Nedah Jan Safi

[x20347946](#)

x20347946@student.ncirl.ie

Safe Scanner

Technical Report

## Contents

Executive Summary .....	3
1.0 Introduction .....	3
1.1. Background .....	3
1.2. Aims.....	4
1.3. Technology .....	5
1.4. Structure .....	5
2.0 System.....	6
2.1. Requirements.....	6
2.1.1. Functional Requirements.....	6
2.1.1.1. Use Case Diagram .....	6
2.1.1.2. Requirement 1 <b>QR Code Scanning</b> .....	7
Description & Priority.....	7
Use Case .....	7
2.1.1.3. Requirement 2 <b>Enter URL for Scanning</b> .....	9
Description & Priority.....	9
Use Case .....	9
2.1.1.4. Requirement 3 Image selection for scanning .....	11
Description & Priority.....	11
Use Case .....	11
2.1.2. Data Requirements .....	13
2.1.3. User Requirements .....	14
2.1.4. Environmental Requirements .....	14
2.1.5. Usability Requirements.....	14
2.2. Design & Architecture .....	14
2.3. Implementation .....	18
2.4. Graphical User Interface (GUI).....	26
2.5. Flow Diagram .....	33
2.6. Testing.....	34
2.7. Evaluation .....	43
3.0 Conclusions .....	45
4.0 Further Development or Research .....	46
5.0 References .....	47
6.0 Appendices.....	47
6.1. Project Proposal .....	47
7.0 Objectives.....	49

7.1.	Reflective Journals .....	55
12.1.	Other materials used .....	61

## Executive Summary

The purpose of creating Safe Scanner a web application is to enhance user security by ensuring that scanned QR codes lead to legitimate and safe URLs. Detecting malicious URLs is crucial for network and cybersecurity. Malicious URLs pose a significant threat, sharing unsolicited content like spam, phishing, and drive-by downloads. They lure unsuspecting users into scams, leading to monetary loss, theft of private information, and malware installation. (arxiv, 2019). For detecting malicious URLs blacklists have been the standard method for detecting harmful URLs, but they are not always accurate and fail to identify newly created malicious URLs. In recent years, machine learning approaches have attracted attention as an approach of improving the reliability of malicious URL detectors. Detecting malicious URLs using machine learning techniques. Machine learning techniques have been increasingly applied to solve the problems relating to information security and cybersecurity. Malicious URL (Uniform Resource Locator) detection is one of these (Vanhoenshoven et al. 2016). In this project, I've applied the random forest method to develop a machine learning model incorporating lexical features, host-based features, and content-based features. The model has an accuracy of 94.7%. The Safe Scanner web application operates in real-time, capturing video feed for QR code detection and processing. The code employs a systematic approach, checking for URL validity, legitimacy, and potential threats. The application is developed using python, JavaScript, HTML, CSS, Flask framework and VS Code. The report highlights the code implementation, emphasizing the integration steps. Users are provided with informative outputs regarding the safety and legitimacy of the detected QR code's URL. The system is flexible, allowing for future enhancements such as additional threat checks and improved user interfaces for a more intuitive experience.

## 1.0 Introduction

### 1.1. Background

In the 21st century, technology permeates every aspect of our lives, providing solutions that make tasks easier and more efficient. QR codes have emerged as a popular means of sharing information rapidly, easily detectable by scanners. While QR codes are created in way that they are not inherently readable by end-users, their information can only be retrieved by a scanning device. The simplicity of scanning QR codes to access URLs has made it a prevalent practice, eliminating the need for users to manually input lengthy web addresses. However, this convenience introduces a significant challenge – the ease with which threat actors can manipulate QR codes to disseminate false or malicious URLs. Malicious actors exploit this vulnerability by automating URL creation and implementing redirects, trapping unsuspecting users without their knowledge. In response to this growing security concern, I am developing Safe Scanner a web application to scan QR codes, assess their content, and determine whether they contain URLs. If a URL is present, the application will further analyse its legitimacy, helping users distinguish between safe, clean URLs and potentially harmful or malicious ones.

The initiative project aims to empower users with a tool that enhances their awareness and security when interacting with QR codes. By scrutinising the URLs embedded in QR codes, the application serves as a preventive measure against falling victim to malicious schemes. This project aligns with the evolving landscape of technology, where ensuring the security of everyday tools like Safe Scanner QR code becomes imperative to foster a safe and trustworthy digital environment. To get

the above using machine learning algorithms such as Random Forest. Random forest is an ensemble learning method for combining the outputs from several decision trees to come up with a final prediction. Each of the decision trees is built on a random subset of the features and on a random subset of the training data to reduce overfitting and improve model robustness. In random forest, the classification model will be built using a dataset of URLs marked as malicious or benign. This model will learn many features of URLs, including lexical, host-based, and content-based features. The model will then be used to classify unknown URLs as malicious or benign. Once the random forest model is given a new URL for which the malicious status is required, each of the individual decision trees is used to predict the malicious status for that URL, based on the features that the URL presents. The predictions made by the individual trees are combined to come up with the final prediction. The random forest algorithm combines the predictions of many decision trees, which decreases variance. To give more reliable predictions for known and unknown malicious URLs. The use of random forest model, which detects malicious URLs, has the advantage of reducing false outcomes. False positives occur when benign URLs are misclassified as malicious, Resulting in unnecessary blocks or warnings. The Random Forest can significantly reduce false Positives enhance overall accuracy by providing a strong and accurate classification of the outcomes.

The URLs are obtained in a dataset from Kaggle, the dataset contains about 100,000 URLs where 70% of those are benign and 30% are malicious. The next step after fetching the dataset is data cleaning. Change the texted data to numeric. Extract the necessary features of URL, these features are extracted from a URL to classify it as either malicious or not. The features we have used are further divided into lexical, host-based, and content-based features. The approach applies lexical features extracted from the URL string to detect attacks based on distinguishable visual characteristics of malicious and benign URLs through statistical analysis.

We then deployed the Random Forest algorithm, a popular ML algorithm commonly used for classification tasks. The Random Forest algorithm trains a classification model on a dataset comprising URLs labelled as either malicious or benign. A total of 22 features were considered for training our model, including lexical, host, and content-based features. These features were then extracted from the URLs using a variety of techniques like regular expressions, domain analysis, and webpage content analysis. One of the issues we encountered during the training of our model was the class imbalance issue, where the number of malicious URLs was much fewer than the benign ones.

## 1.2. Aims

The primary aim of this project is to create a QR code scanner application that not only detects QR codes but also validates the associated URLs. Specifically, the application aims to achieve the following:

- Accurate detection and decoding of QR codes from video feeds or images.
- Develop a system capable of scanning QR codes and extracting the content.
- Verification of the extracted content to determine if it constitutes a valid URL.
- Assessment of the legitimacy of the URL.
- Enable flexibility, by letting the user to detect QR code and URLs.
- Developed a machine learning-based system to classify URLs into different categories.
- train model that accurately predicts whether a URL is malicious or benign.

- Developed a real-time URL detection server using Flask, allowing clients to test URLs for potential threats.

### 1.3. Technology

The project utilizes the Python programming language, Python is well-suited for machine learning tasks due to its extensive ecosystem of libraries and ease of use. JavaScript: JavaScript was employed to implement the QR code scanning logic, content type analysis, enhance user interaction and enable real-time URL detection, HTML and CSS to structure and style the user interface. Flask framework: Flask allows for quick setup of web servers and provides the flexibility needed for integrating machine learning models. VS Code: Chosen for its support of multiple languages, rich extensions, and built-in debugging capabilities. It offers a robust development environment for both frontend and backend code. Machine learning algorithms implemented using scikit-learn and TensorFlow to enhance QR code analysis and threat detection based on learned patterns.

### 1.4. Structure

This document is structured to provide a comprehensive overview of the Safe Scanner application. It includes sections detailing the system architecture, code implementation, and optional enhancements such as additional threat checks. The executive summary encapsulates the key points of the report, offering a succinct overview of the project's purpose, major components, and potential future developments. The subsequent sections delve into each aspect of the project, providing technical insights and guidance for developers interested in similar endeavours. The following are the brief overview and document structure and content in each section.

- 1.0 introduction: this section is the foundation section of the document. It provides an overview of the project, its scope, and objectives of the application.
- 2.0 System: this section of the document describes the architecture of system and provide details about the requirements.
- 3.0 Fundamental Requirements: this section of the document describes the key functionalities that the project supports such as QR code scanning, URL detection, content type, and error handling.
- 4.0 Non-Functional requirements: this section of the document describes the non-functional requirements of the Safe Scanner application, e.g. user interface, performance, reliability, and security.
- 5.0 Use Cases: the section provides the use case diagram of the application, and the textual description of the interaction between the application actors, to achieve project goal.
- 6.0 Design and Architecture: this section of the document explores the design and architecture of the project. It explains how the system is structured and the detail about its component interaction.
- 7.0 Implementation: this section of the document describes the code and key algorithms used in the project, it explains how the system is build and highlight interesting code snippet.
- 8.0 Testing and Evaluation: this section of the document focuses on the testing and evaluating the project, it explains how the system was testing and it result.

9.0 Conclusion: this section of the document provides a summary of the document, summaries the funding and outcomes of the project.

## 2.0 System

### 2.1. Requirements

The system requirements for the QR code Safe Scanner application with integrated URL validation and threat checking are outlined to ensure verifiability, user-friendliness, and effective functionality. These requirements cover aspects such as user training, error tolerance, user roles, and ease of use.

#### 2.1.1. Functional Requirements

The functional requirements outline the key capabilities the system must possess to provide a secure, efficient, and user-friendly QR code scanner application with integrated URL validation and threat checking.

1. QR Code Scanning: the system must be able to scan QR code and extract its content.
2. Real-Time Scanning: The QR code scanner should operate in real-time, which must provide immediate result to users upon successful scanning.
3. Error Handling: The scanner must detect and handle errors gracefully, providing informative error messages to users and allowing for recovery from issues.
4. Content Detection: The system must be able to identify the type of content extracted from the QR code. This includes distinguishing between URLs, phone numbers, email addresses, and plain text.
5. Regular Expression Matching: The content analysis must use regular expressions or similar pattern-matching techniques to accurately classify different content types.
6. Content Type Handling: Based on the detected content type, the system should take appropriate action (e.g., classify URLs, create clickable links for phone numbers, etc.).
7. URL Feature Extraction: The system must analyse URLs to extract various lexical features, including length, number of special characters, redirect count, digit count, entropy, and others.
8. Suspicious Pattern Detection: The system must check for specific patterns or characteristics that could indicate a suspicious or malicious URL, such as suspicious keywords, unusual TLDs, or IP addresses in the hostname.
9. Benign Pattern Detection: The system should also identify benign patterns, such as common TLDs, benign keywords, and limited redirects.
10. Output: The system must present results in a clear and user-friendly manner, allowing users to understand the outcome of the scan.

##### 2.1.1.1. Use Case Diagram

A use case diagram is a visual representation that illustrates the interactions between users and a system, showcasing various use cases and their relationships. In the context

of the Safe Scanner application with integrated URL validation and threat checking, the following use case diagram outlines the primary interactions:

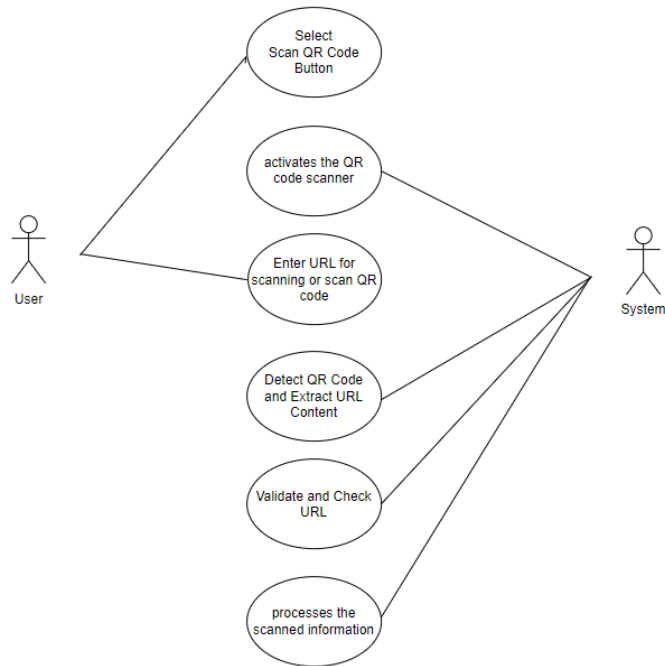


Figure 1 Use Case Diagram

2.1.1.2. Requirement 1 QR Code Scanning
<b>Description &amp; Priority</b> <p>This requirement involves the functionality of opening the Safe Scanner web app, selecting the "Scan QR code" button, then scanning the QR code. This is a critical requirement as it forms the core functionality of the application.</p>
<b>Use Case</b> <p><b>Scope</b></p> <p>The scope of this use case is to initiate and perform the QR code scanning process, either by selecting the "Scan QR code" button or manually entering a URL.</p> <p><b>Description</b></p> <p>This use case describes the steps involved in opening the QR code scan app, selecting the appropriate option for scanning, and performing the actual QR code scan.</p>



## Use Case Diagram

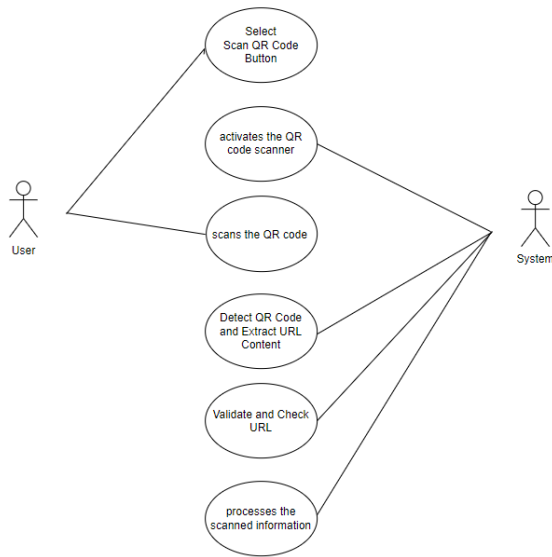


Figure 2 Use Case Diagram of Scan QR Code

## Flow Description

### Precondition

The system is in initialization mode.

### Activation

This use case starts when a user initiates the QR code scanning process by opening the app.

### Main flow

1. The system identifies the user's choice to either "Scan QR code" or "Enter URL for Scanning."
2. If the user selects "Scan QR code," the system activates the QR code scanner.
3. The user scans the QR code using the device's camera.
4. The system processes the scanned information.

### Alternate flow

A1: Scan QR Code Button Not Selected

1. The system prompts the user to select a scanning method.
2. The user selects "Enter URL for Scanning."
3. The use case continues at position 4 of the main flow.

<p><b>Exceptional flow</b></p> <p>E1: QR Code Scan Failure</p> <ol style="list-style-type: none"> <li>1. If the system encounters difficulties in scanning the QR code:</li> <li>2. The system notifies the user.</li> <li>3. The use case continues at position 4 of the main flow.</li> </ol>
<p><b>Termination</b></p> <p>The system presents the next relevant action based on the scanned information.</p>
<p><b>Post condition.</b></p> <p>The system goes into a wait state.</p>

<p><b>2.1.1.3. Requirement 2 Enter URL for Scanning</b></p>
<p><b>Description &amp; Priority</b></p> <p>This requirement involves allowing the user to enter a URL for scanning in the Safe Scanner web app. It is essential for users who prefer entering the URL manually. This functionality is considered critical for the overall system.</p>
<p><b>Use Case</b></p>
<p><b>Scope</b></p> <p>The scope of this use case is to facilitate the manual entry of a URL for scanning within the QR code scanning app.</p>
<p><b>Description</b></p> <p>This use case outlines the steps involved when a user chooses to manually enter a URL for scanning instead of scanning a QR code.</p>

## Use Case Diagram

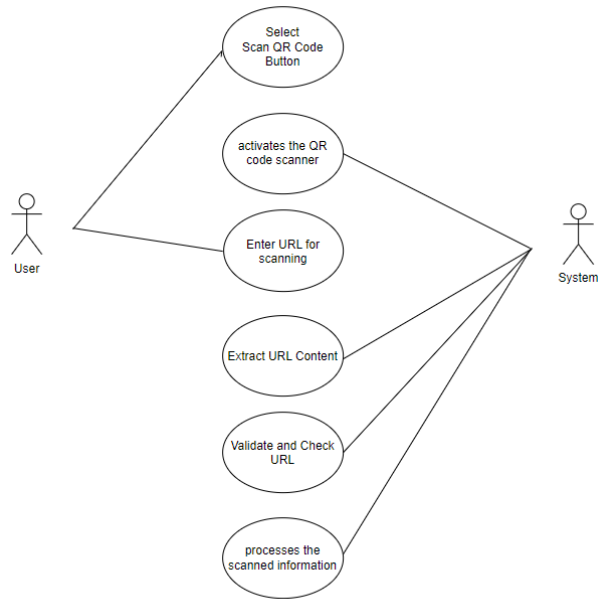


Figure 3 Use Case Diagram of URL

### Flow Description

#### Precondition

The system is in initialization mode.

#### Activation

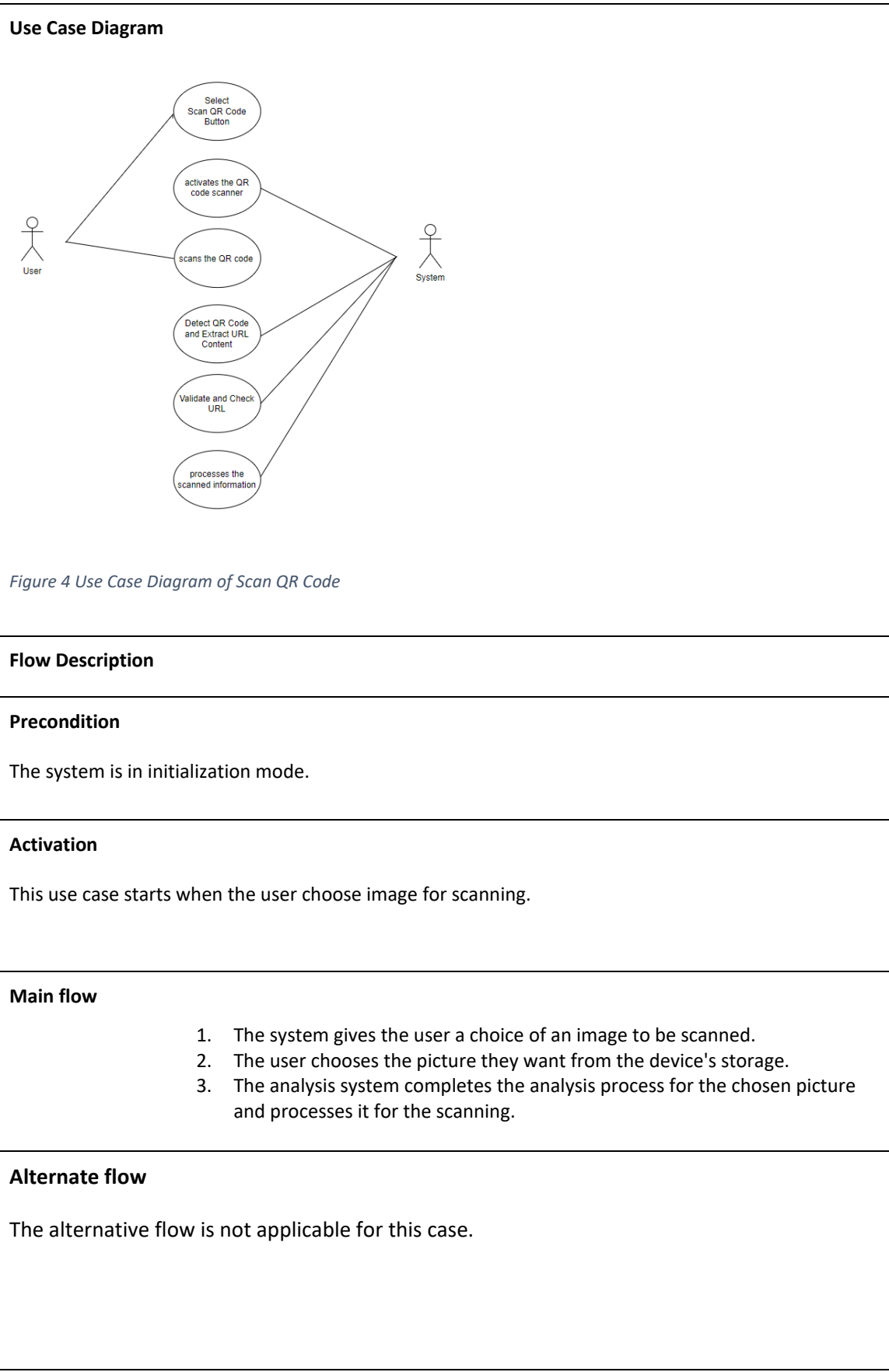
This use case starts when a user initiates the QR code scanning process by opening the app and selecting the option to enter a URL.

#### Main flow

1. The system identifies the user's choice to enter a URL for scanning.
2. The user manually enters the URL into the designated input field.
3. The system processes the entered URL.

<p><b>Alternate flow</b></p> <p>A1: Scan QR Code Button Selected</p> <ol style="list-style-type: none"> <li>1. The system prompts the user to select a scanning method.</li> <li>2. The user selects "Scan QR code."</li> <li>3. The use case continues at position 2 of the main flow.</li> </ol>
<p><b>Exceptional flow</b></p> <p>E1: Invalid URL Entry</p> <ol style="list-style-type: none"> <li>1. If the user enters an invalid or malformed URL:</li> <li>2. The system notifies the user about the error.</li> <li>3. The use case continues at position 2 of the main flow.</li> </ol>
<p><b>Termination</b></p> <p>The system presents the next relevant action based on the entered URL.</p>
<p><b>Post condition.</b></p> <p>The system goes into a wait state.</p>

<p>2.1.1.4. Requirement 3 Image selection for scanning</p>
<p><b>Description &amp; Priority</b></p> <p>This requirement involves the functionality of open the Safe Scanner web app, select the "Scan QR code" button, then select "Scan Image File". This functionality of the application usability.</p>
<p><b>Use Case</b></p>
<p><b>Scope</b></p> <p>The scope here is that the user will have an option to select an image from their device's storage to feed it for the scanning.</p>
<p><b>Description</b></p> <p>Description The next use case will show the steps of selecting an image from the device's storage within the "Safe Scanner" app and starting the scanning.</p>



<p><b>Exceptional flow</b></p> <p>E1: The selected image is not a QR Code image.</p> <ol style="list-style-type: none"> <li>4. If the system encounters difficulties in accessing the device's storage or retrieving the selected image:</li> <li>5. The system alerts the user on the problem.</li> <li>6. The use case continues at position 2 of the main flow.</li> </ol>
<p><b>Termination</b></p> <p>The system responds by displaying the appropriate next action or goes back to the previous condition</p>
<p><b>Post condition.</b></p> <p>The system is still in a state that allows the user to continue interacting with it or it switches back to the initialization mode if the user does nothing.</p>

### 2.1.2. Data Requirements

The Data Requirements for the Safe Scanner Application with Integrated URL Validation and Threat Checking demonstrate what the system needs to carry out the functions it was designed to perform. This need includes several data sets and inputs, which are imperative to correctly coping with QR codes and link checking.

1. **QR Code Data:** The text that is extracted from the QR code, which is supposed to be a URL.
2. **QR Code Content:** The information contained in a QR code. Can be a URL, or other text.
3. **QR Code Format:** QR codes can encode different types of data, the project must be able to read all kind of formats.
4. **URL Components:** The URL format, which comprises the protocol (http/https), hostname, path, query parameters, and port.
5. **Lexical Patterns:** Some of the patterns for identifying these special characters, digits, and other lexical features within the URL.
6. **Suspicious Indicators:** It is suggested to intervene into various URLs keywords and patterns which contain words that would warn users about possible phishing or malicious URLs, like confusing TLDs.
7. **Benign Indicators:** Also Benign URL is denoted by the substantive keywords and other signs, for example .com or .i.e. can help to track down relevant articles by restricting search queries, as well as keywords that denote safe content (e. g. , "home", "about").
8. **Scanner Configuration:** Information about the QR code scanner's settings, like the number of frames per second (fps), scanning box size, and error handling.
9. **HTTP Response Status Code:** Whether a URL is legit or not is the application's criterion, and it checks it with the 200-status code of HTTP response meaning OK.
10. **Machine Learning Model Parameters:** Machine learning algorithm setting up parameters and weights.

### 2.1.3. User Requirements

The user requirements for Safe Scanner application including integrated URL validation and threat checking are identified to provide an easy, fast, and secure experience for users. These necessities are established from understanding of expectations and requirements of the end-users of the app.

- **Real-time QR Code Detection:** Users want the application to detect QR codes in real-time from a video feed.
- **URL Information:** The audience wants to know in relation to the URL whether it's legitimate and secure or not.
- **Intuitive Output:** The application should be designed in such a way that it will render clear and precise output, expressing whether the URL is malicious, or it is safe to browse upon.
- **Optional Features:** The users may like additional features, such as, for example, fetching additional data for more detailed URL analysis.

### 2.1.4. Environmental Requirements

- **Camera Access:** Application using a camera for QR scanning need to have a camera enabled to facilitate the actual image capturing.
- **Internet Connectivity:** It is necessary to perform URL legitimacy and threat checks through the system as this is a web application. Therefore, the system needs an internet connection to be able to operate.
- **Hardware Compatibility:** The software must be able to run on camera devices based on ordinary hardware configurations, of the kind that are ubiquitous.

### 2.1.5. Usability Requirements

- **User-Friendly Interface:** The application has a simple and easy-to-use interface.
- **Prompt Response:** fast detection and analysis of the QR code, and the immediate response to it.
- **Error Handling:** The application is capable of handling errors, and in case there is a request failure or camera access problems, the application will provide informative messages.
- **Configurability:** The program can let the users to set up some parameters, for example, changing the camera settings.

## 2.2. Design & Architecture

Design and architecture of this project aim at integrating machine learning into a system, which ensures the reliability, scalability, and maintainability during the identification of malicious URLs. The system is built up on the principle of 'end-to-end workflow' that contains all the data preprocessing and the training, deployment, and evaluation of the machine learning algorithm. The technology allows for real-time URL identification online using a web-based interface that connects the frontend and backend technologies.

- **Frontend Interface:** This is a web-based interface, which makes it possible to scan the QR codes.

- QR Code Scanner: This Html5QrcodeScanner reads QR Codes and, if reader has successfully scanned the barcode, it invokes the callback function.
- Content Type Analysis: The technique used to identify which kind of content is retrieved from the QR code.
- URL Feature Extraction: A feature that uses the lexical features to analyses the URL.
- Result Display: The frontend displays to the user the feedback of the scanned content and the results of URL analysis.
- Data Layer: Manages data collection, preparation, and storage. The second layer of the process will then deal with the data collected from the external sources to map it for the machine learning model.
- Model Layer: Containing a learning logic for training and evaluation of machine learning algorithms. This layer uses pre-processed data to train the model and evaluates its performance.
- The application layer involves business logic for real-time URL detection and communication with the model of machine learning. The server-side communication points are provided using Flask frameworks.
- The Presentation Layer is the layer that allows the user to interact with the system through a user-friendly interface. This layer is responsible for creating a responsive frontend by combining HTML, CSS, and JavaScript.

To detect malicious QR code, the process was divided into three stages. The first stage is the stage when the QR code is scanned using the “Html5QrcodeScanner” library to scan QR Code. The Html5QrcodeScanner is responsible for creating a QR code scanner. One which permits to read QR codes. After the QR code is scanned, the “urlPattern. test” function is invoked to detect the QR code types. The machine learning model is used for scanning the URL in the second stage. Feature classification is the 3rd stage in frame work which is composed of several classes, the first Item is the Lexical feature extraction from URL to find the suspicious URLs, the 3rd stage of frame work is to evaluate the value that giving by the features and evaluate the results to find out if the URL is malicious or not then if the features have provided the value it will move to final computation and sending the result to user.



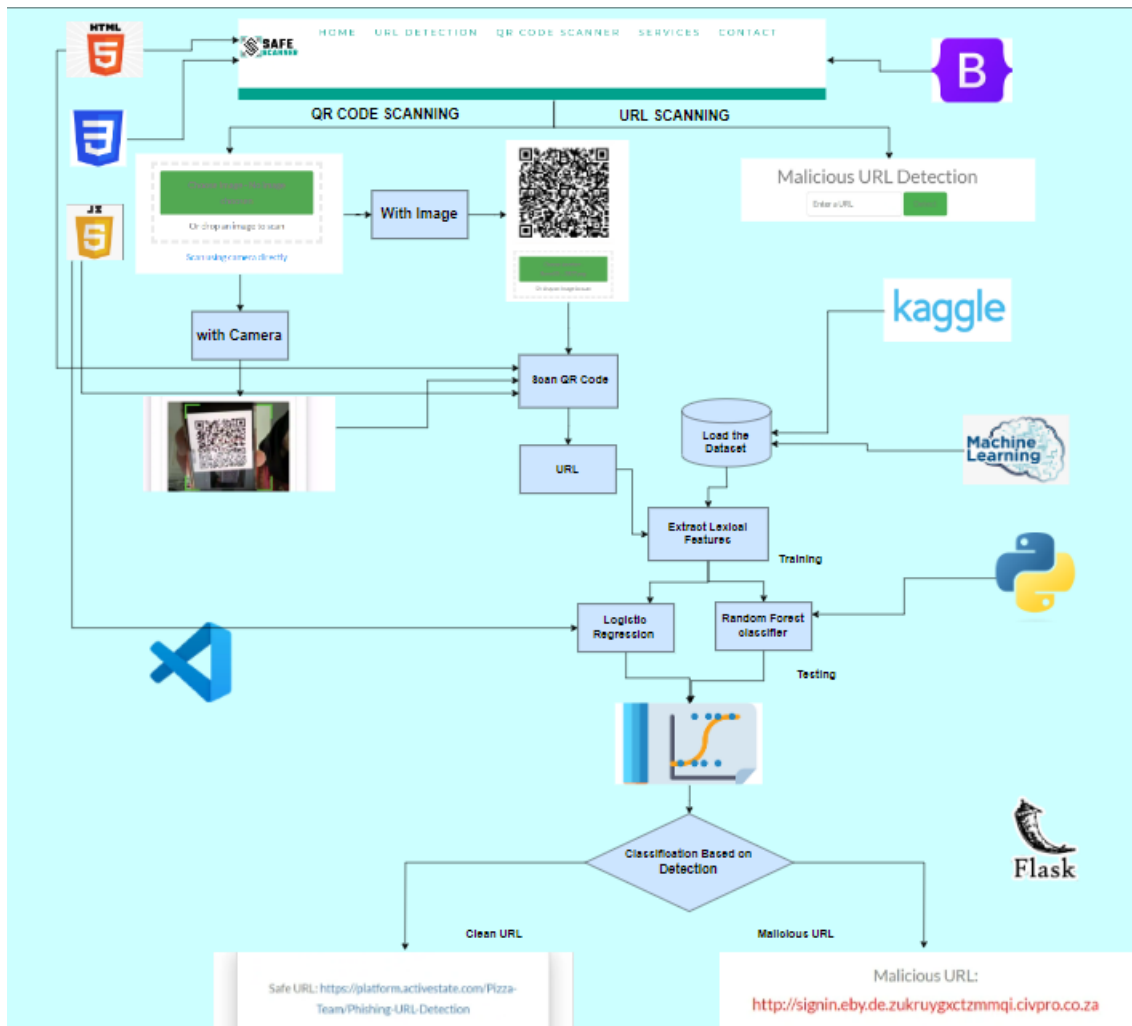


Figure 5 Architecture diagram.

This architecture diagram highlights information flow through different modules from QR-code detection to URL validation, legitimacy check and advanced machine learning algorithms. Each module of the software plays its part to form an overall application that is functional and secure.

No	Classes	Reasons	Type
1	%	Used to identify URL HTML encoding, existence of IDs (i.e., Session IDs, Affiliate IDs, Tracking IDs, Referrer IDs) and time stamps.	Numeric
2	//	It is used to identify segment part.	Numeric
3	/	It is used to compute path's length in the URL.	Numeric
4	.	It is used to determines the relative references within path hierarchy, domain name, SLD, and TLD.	Numeric
5	=	It is used to identify assignment of values in URL.	Numeric
6	-	It is used to count number of word joiners.	Numeric
7	_	It is used to count number of word separators.	Numeric
8	@	It used to extract and analyzed whether any user information exists in the URL or not.	Numeric
9	?	It is used to identify the query section of the URL, if any.	Numeric
10	:	It has been used to determine whether a port is in use.	Numeric
11	~	It has been used to determine whether the URL has any reference or not.	Numeric
12	URL Length	It has been used to identify the obfuscation technique mostly employed in phishing attacks.	Numeric
13	Hostname Length	It used to identify the obfuscation technique mostly employed in phishing attacks.	Numeric
14	Path Length	It used to identify the obfuscation technique mostly employed in phishing attacks.	Numeric
15	Count Redirect	It used to identify the obfuscation technique, it mostly employed to pass the detection models. This information of this class gathered from preprocessing phase.	Numeric
16	Count Digits	It used to count the number of digits has been used in the URL.	Numeric
17	Exist IP	Some malware hosting websites don't have domain names and are instead identified by their IP addresses. Mostly used for drive by download attack.	Binary
18	Exist Port	Using certain ports creates a connection between the web browser and the web servers, exposing sensitive user data to cybercriminals and potentially resulting in severe data misuse.	Binary
19	Absolute URL	An absolute URL contains more information from protocol to TLD. However, a relative URL does not use the full web address and only contains the location following the domain name and is mostly used for obfuscation purposes.	Binary
20	Suspicious TLD	The lengthy list of suspicious top-level domains with 2 levels of high risk and mid-risk is mostly used for phishing and malware attacks.	Binary
21	Suspicious Words	The lengthy list of suspicious words has 2 levels of high risk and mid-risk, which are mostly used for phishing and malware attacks.	Binary
22	File Extension	It has been mostly used by attackers to launch a drive-by-download attack on their victims by downloading executive files. It is a list of file extensions that includes all the executive files running in different operating systems.	Binary

Figure 6 lexical features of URL.

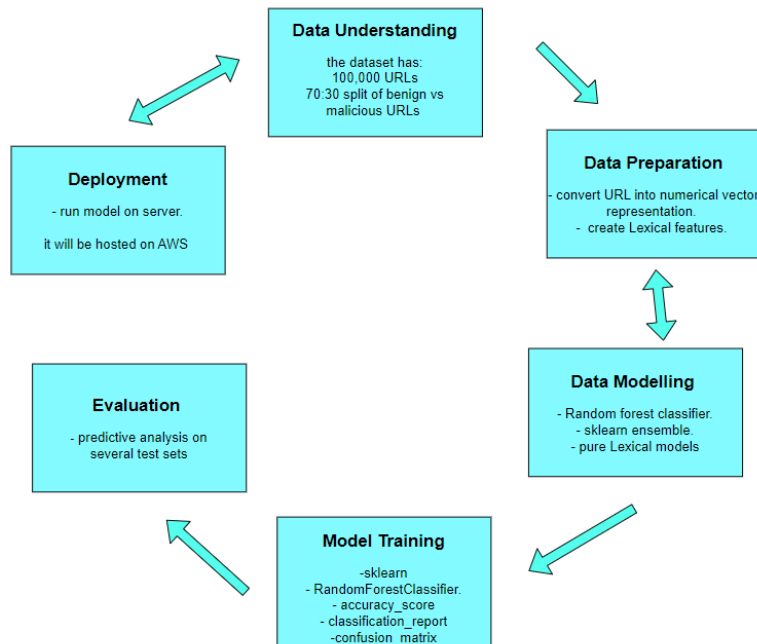


Figure 7 machine learning model main steps.

## 2.3. Implementation

The Safe Scanner application, in the implementation section, gives the details of the main algorithm, classes, and functions that are used in the project. The manual is concise and provides a clear understanding of the critical components.

The makeup of my code from mid-point submission have been changed slightly, after research and looking truth of a project, I chose to make app-based web instead of a mobile application. To facilitate the user to get access. The denial of API use is another aspect of this assignment that it could learn as API cannot be responsible of newly developed malicious URLs, even I want to detect all URLs by machine learning.

**data\_preprocessing.py:**

The screenshot shows a PyCharm IDE window titled "SafeScannerApp". On the left, there's a sidebar with icons for Explorer, Search, Run and Debug, and Outline. The main editor area displays a file explorer on the left pane showing a project structure under "SAFESCANNERAPP" with files like .venv, dataset, static, templates, .env, .gitignore, data\_preprocessing.py, index.html, QR\_Logo.png, qrScanner.css, qrScanner.html, qrScanner.js, README.md, requirement.txt, scaler.pkl, styles.css, train\_model.py, url\_dataset.csv, url\_detection\_model.pkl, url\_detection.py, url\_scaler.pkl, urls\_data.csv, urScanner.css, urScanner.html, urScanner.js, X\_test\_scaled.pkl, X\_train\_scaled.pkl, y\_test.pkl, and y\_train.pkl. The right pane shows the code for "data\_preprocessing.py":

```
1 import pandas as pd
2 import numpy as np
3 from sklearn.model_selection import train_test_split
4 from sklearn.preprocessing import StandardScaler
5 import joblib
6
7 # Load a dataset with URLs and labels
8 # The dataset with a 'url' column and a 'label' column
9 url_data = pd.read_csv("urls_data.csv")
10
11 # Define the correct Label mapping based on the dataset's unique values
12 label_mapping = {
13     "benign": 0,
14     "defacement": 1,
15     "phishing": 2,
16     "malware": 3,
17 }
18 # Handle unexpected labels by providing a default value
19 0: "benign", # Example of handling numerical labels
20 1: "defacement",
21 2: "phishing",
22 3: "malware"
23 }
24
25 # Apply the corrected label mapping
26 url_data['label'] = url_data['label'].apply(lambda x: label_mapping.get(x, 0))
27
28 # Function to extract lexical features from URLs
29 def extract_features(url):
30     return {
31         "url_length": len(url),
32         "num_special_chars": len([char for char in "%&.-:_"]),
33         "num_digits": sum(char.isdigit() for char in url),
34         "contains_ip": any(char.isdigit() for char in url.split('.')),
35         "contains_special": any(char in ['%', '/', '.', '=', '-', '@', '?', ':', '-'] for char in url)
36     }
```

At the bottom status bar, it indicates "Ln 1, Col 10 Spaces: 4 UTF-8 CRLF Python 3.11.0.2 64-bit".

Figure 8 data\_preprocessing.py

The picture above portrays the data\_preprocessing. process.py which is the file that consists of functions for data preprocessing dedicated to machine learning. This is a multistage process with such information such as data sorting, data feature extraction and data labelling converting. Figure 9

### Label Mapping:

```

data_preprocessing.py > ...
7
8 # Load a dataset with URLs and labels
9 # The dataset with a 'url' column and a 'label' column
10 url_data = pd.read_csv("urls_data.csv")
11
12 # Define the correct label mapping based on the dataset's unique values
13 label_mapping = {
14     "benign": 0,
15     "defacement": 1,
16     "phishing": 2,
17     "malware": 3,
18     # Handle unexpected labels by providing a default value
19     0: "benign", # Example of handling numerical labels
20     1: "defacement",
21     2: "phishing",
22     3: "malware"
23 }
24
25 # Apply the corrected label mapping
26 url_data['label'] = url_data['label'].apply(lambda x: label_mapping.get(x, 0))
27
28

```

Figure 9 label mapping.

The picture above demonstrates that the feature “label\_mapping” was applied to the text labels to transform them to numeric values for the machine learning. Figure 10

```

28
29 # Function to extract lexical features from URLs
30 def extract_features(url):
31     return {
32         "url_length": len(url),
33         "num_special_chars": len([char for char in "%/._-:"]),
34         "num_digits": sum(char.isdigit() for char in url),
35         "contains_ip": any(char.isdigit() for char in url.split('.')),
36         "contains_special": any(char in ['%', '/', '.', '=', '_', '-', '@', '?', ':', '~'] for char in url)
37     }
38
39 # Create a DataFrame with the extracted features
40 feature_data = pd.DataFrame([extract_features(url) for url in url_data['url']])
41
42 # Add the labels to the features DataFrame
43 feature_data['label'] = url_data['label']
44
45 # Split the data into training and testing sets
46 X_train, X_test, y_train, y_test = train_test_split(
47     feature_data.drop("label", axis=1),
48     feature_data['label'],
49     test_size=0.3,
50     random_state=42
51 )
52
53 # Standardize the features
54 scaler = StandardScaler()
55 X_train_scaled = scaler.fit_transform(X_train)
56 X_test_scaled = scaler.transform(X_test)
57

```

Figure 10 extract URL feature.

The “extract\_features” works on creating numerical representations of the URLs so that they can be used in machine learning. “StandardScaler” is used to scale down the data and ensure the features of all the things are in a similar scale. It’s the scaling of the features to standard deviation to enhance the performance of machine learning algorithms. Figure 11

## Model Training:

```
train_model.py 5, U x
train_model.py > ...
1 from sklearn.preprocessing import StandardScaler
2 from sklearn.ensemble import RandomForestClassifier
3 from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
4 import pandas as pd
5 import joblib
6
7 # Load the data from the data_preprocessing step
8 # (Ensure this file runs after 'data_preprocessing.py')
9 X_train_scaled = pd.read_pickle("X_train_scaled.pkl")
10 X_test_scaled = pd.read_pickle("X_test_scaled.pkl")
11 y_train = pd.read_pickle("y_train.pkl")
12 y_test = pd.read_pickle("y_test.pkl")
13
14 # Train a Random Forest model
15 model = RandomForestClassifier(n_estimators=100, random_state=42)
16 model.fit(X_train_scaled, y_train)
17
18 # Evaluate the model
19 y_pred = model.predict(X_test_scaled)
20
21 # Calculate evaluation metrics
22 accuracy = accuracy_score(y_test, y_pred)
23 classification_rep = classification_report(y_test, y_pred)
24 confusion_mat = confusion_matrix(y_test, y_pred)
25
26 # Display the evaluation results
27 print("Accuracy:", accuracy)
28 print("Classification Report:")
29 print(classification_rep)
30 print("Confusion Matrix:")
31 print(confusion_mat)
32
33 # Save the trained model and scaler for future use
34 scaler = joblib.load("url_scaler.pkl")
35
36 joblib.dump(model, "url_detection_model.pkl")
37 joblib.dump(scaler, "url_scaler.pkl")
```

Figure 11 model taking.

The model training which is the process of designing the machine learning algorithms to categorize the URLs to different groups. The system takes real-time data provided by the Wearable Sensor Network and makes use of the popular Random Forest classifier machine learning algorithm for predictive classifications. Hierarchical Random Forest classifier is selected as it can robustly handle with multi-feature integration as well. Figure 12

```
13
14 # Train a Random Forest model
15 model = RandomForestClassifier(n_estimators=100, random_state=42)
16 model.fit(X_train_scaled, y_train)
17
```

Figure 12 Train Random Forest model.

“RandomForestClassifier” function inserts a training process for the Random Forest Classifier model. The parameter “n\_estimators=100” is the tree number that the model will use in the forest “random\_state=42” that is used to make the model reproducible. Figure 13

## Flask Server

### url\_detection.py

```
url_detection.py 4, U x
url_detection.py > ...
1  from flask import Flask, request, jsonify, send_file
2  from flask import Flask, render_template
3  import joblib
4  import re
5  import pandas as pd
6
7  app = Flask(__name__)
8
9  # Load the trained model and scaler
10 model = joblib.load("url_detection_model.pkl")
11 scaler = joblib.load("url_scaler.pkl")
12
13 # Endpoint to serve the HTML file
14 # Route to the home page
15 @app.route("/")
16 def index():
17     return render_template("index.html")
18
19 # Route to the about page
20 @app.route("/qrScanner")
21 def about():
22     return render_template("qrScanner.html")
23
24
25 # Endpoint to serve the HTML file
26 @app.route("/")
27 def home():
28     return send_file("urlScanner.html") # Serve the HTML file
29
30 # Endpoint for URL detection
31 @app.route("/detect", methods=["POST"])
32 def detect_url():
33     data = request.json
34     url = data.get("url", "")
35
36     # Extract features from the URL
```

Figure 13 Flask server.

```
44     # Standardize the features
45     features_scaled = scaler.transform(pd.DataFrame([features]))
46
47     # Predict whether the URL is suspicious or benign
48     prediction = model.predict(features_scaled)[0]
49
50     result = {
51         "url": url,
52         "is_suspicious": bool(prediction),
53         "message": "Malicious URL" if prediction else "Benign URL",
54     }
55
56     return jsonify(result)
57
58 # Run the Flask app
59 if __name__ == "__main__":
60     #app.run(debug=True, port=5000)
61     app.run(debug=True, host='0.0.0.0', port=8080)
62
```

Figure 14 Flask server connection with frontend.

The flask server has been configured properly to accept the data from the end point and send the prediction. Figure 15

A flask-based server is created to help detect the URLs in real time. Here we will be using the “joblib.load” function to load the trained model and the scaler. The “route” function is employed for html rendering and endpoint files. The server is the one that will make the POST requests to get the URL data and then it will use the machine learning trained model to detect if the URL is benign or malicious. Figure 14

## qrsScanner.js

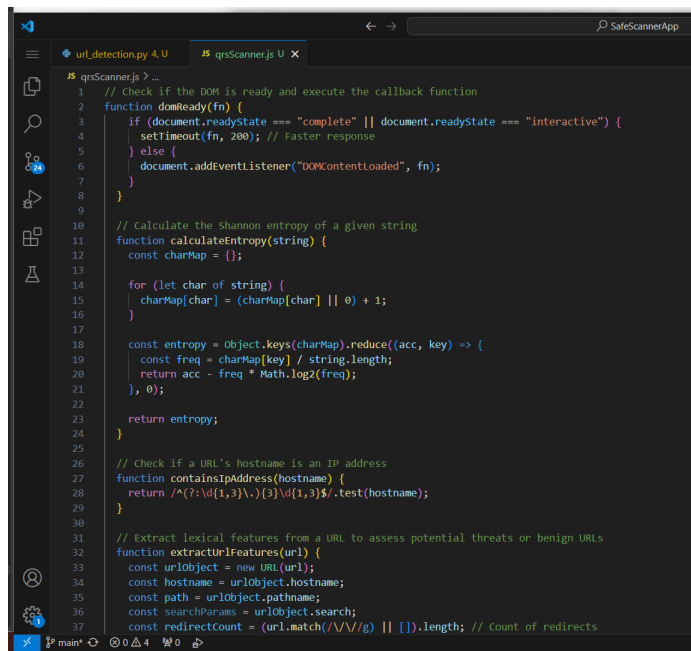


Figure 15 QR Code Scanner.

The above shown is the main page for scanning QR codes and for the analysis of the content and the type of QR codes. Figure 16

## domReady(fn)

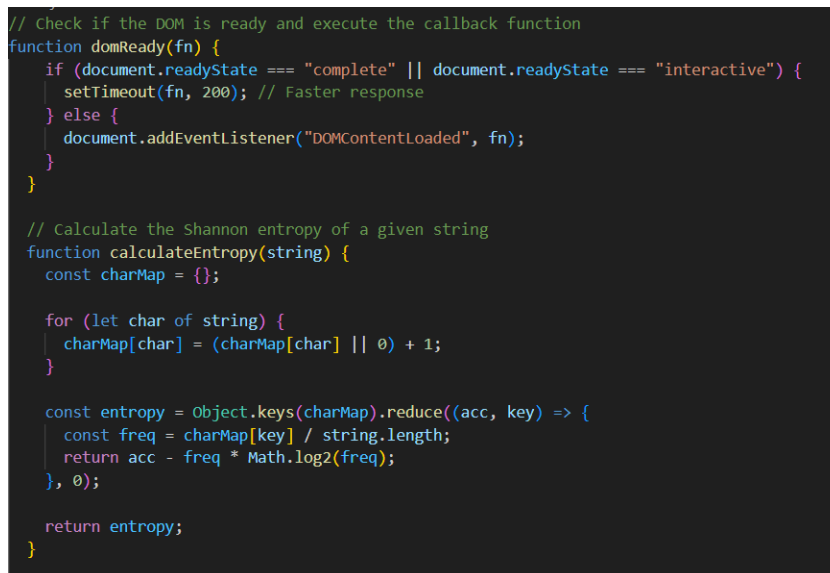


Figure 16 QR Code Scanning.

The above function can be applied while DOM is loading, and a specified callback function is performed. Another thing the dough that itself checks if DOM has already loaded or waits for the event to load. The calculateEntropy(string) function which is the first step to calculating the entropy of a given string is the first step to building a frequency map of characters and finally to calculating the entropy based on the character frequency. Figure 17

## extractUrlFeatures(url)

```
canner.js > calculateEntropy

// Check if a URL's hostname is an IP address
function containsIpAddress(hostname) {
  return /^(?:\d{1,3}\.){3}\d{1,3}$/i.test(hostname);
}

// Extract lexical features from a URL to assess potential threats or benign URLs
function extractUrlFeatures(url) {
  const urlObject = new URL(url);
  const hostname = urlObject.hostname;
  const path = urlObject.pathname;
  const searchParams = urlObject.search;
  const redirectCount = (url.match(/\/\//g) || []).length; // Count of redirects
  const digitsCount = (url.match(/\d/g) || []).length; // Count of digits
  const specialCharsPattern = /[%\V\.\-_\@?~]/g;
  const features = {
    urlLength: url.length, // Total length of the URL
    hostnameLength: hostname.length, // Length of the hostname
    pathLength: path.length, // Length of the URL path
    numSpecialChars: (url.match(specialCharsPattern) || []).length,
    redirectCount, // Number of URL redirects
    digitsCount, // Count of digits
    containsIpAddress: containsIpAddress(hostname), // True if the hostname is an IP address
    containsPort: /\d+$/i.test(hostname), // True if the URL specifies a port
    isAbsolute: url.startsWith("http://") || url.startsWith("https://"), // Absolute URL check
    entropy: calculateEntropy(url),
    containsSuspiciousKeywords: /(login|free|win|prize|click|admin|secure|bonus|deal|offer|cash)/i.test(url),
    hasSuspiciousTld: /(tk|ml|ga|cf|gq)/i.test(hostname), // Common suspicious TLDs
    fileExtension: path.split('.').pop(), // File extension if exists
    isBenignTld: /(com|org|net|edu|gov)/i.test(hostname), // Typical benign TLDs
    containsBenignKeywords: /(home|about|contact|blog|help|faq|support)/i.test(url), // Words in benign URLs
  };

  return features;
}
```

Figure 17 Check URL features.

The above is the main function of the application, the **containsIpAddress(hostname)** checks for the host name if it's an IP address. It uses the regular expression to check if the hostname matches the IPV4 address format. **extractUrlFeatures(url)** this function uses machine learning numerous lexical features from a URL to check whether the URL is malicious or benign. It analyses the URLs length, digit count, number of special characters, entropy, redirect count, and many other attributes, it also checks for the specific patterns that indicate whether a URL is malicious or benign.



## determineContentType(decodeText)

```
// Function called when a QR code is successfully scanned
function onScanSuccess(decodeText, decodeResult) {
  const resultDiv = document.getElementById("output");

  const urlPattern = /^(https?:\/\/)/; // Regex pattern to
  const isUrl = urlPattern.test(decodeText);

  if (isUrl) {
    const features = extractUrlFeatures(decodeText);

    // Define thresholds for determining if a URL is suspicious
    const isSuspicious =
      features.urlLength > 100 ||
      features.hostnameLength > 30 ||
      features.pathLength > 50 ||
      features.numSpecialChars > 10 ||
      features.redirectCount > 1 ||
      features.digitsCount > 10 ||
      features.containsIpAddress ||
      features.containsPort ||
      features.isAbsolute === false ||
      features.entropy > 5 ||
      features.containsSuspiciousKeywords ||
      features.hasSuspiciousTld;
  }
}
```

```
// Define conditions for detecting benign URLs
const isBenign =
  features.urlLength < 50 &&
  features.hostnameLength < 15 &&
  features.pathLength < 20 &&
  features.numSpecialChars < 5 &&
  features.redirectCount === 1 && // Only one redirect indicates a simpler structure
  features.digitsCount < 5 && // Few digits
  features.entropy < 2 &&
  !features.containsIpAddress &&
  !features.containsPort &&
  features.isAbsolute &&
  !features.containsSuspiciousKeywords &&
  features.isBenignTld &&
  features.containsBenignKeywords;

if (isSuspicious) {
  resultDiv.innerHTML = `Malicious URL: <span style="color: red;">${decodeText}</span>`;
} else if (isBenign) {
  resultDiv.innerHTML = `Benign URL: <a href="${decodeText}" target="_blank">${decodeText}</a>`;
} else {
  resultDiv.innerHTML = `Safe URL: <a href="${decodeText}" target="_blank">${decodeText}</a>`;
}
} else {
  resultDiv.innerHTML = `Scanned Text: ${decodeText}`;
}

domReady(function () {
  const html5qrcodeScanner = new Html5QrcodeScanner("my-qr-reader", {
    fps: 10,
    qrbox: 250,
  });
  html5qrcodeScanner.render(onScanSuccess);
});
```

Figure 18 QR code successful scanned.

The above is the main function of the application, the containsIpAddress(hostname) checks for the host name if it's an IP address. The tool uses the regular expression to ensure that the hostname is right in terms of the IPV4 address's format. def extractUrlFeatures(url): this function uses machine learning to extract a multitude of lexical features (urchins, anchors, or tags) from an URL to tell if the couldn't link is malicious or not. It studies the URL's length, digit count, number of special characters, entropy, and redirect count, and many other attributes, it also looks for the specific patterns that show whether a URL is malicious or benign. Figure 19

### URL Scanner:

```
function checkIfMalicious(url) {
  // feature extraction for logistic regression
  const features = {
    hostLength: url.host.length,
    pathLength: url.pathname.length,
    numQueryParams: Array.from(url.searchParams).length,
    hasSuspiciousChars: /[^\a-zA-Z0-9-\.\_]/.test(url.host + url.pathname),
    subdomains: url.host.split('.').length - 5,
  };

  // logistic regression model with hardcoded coefficients and threshold
  const coefficients = {
    hostLength: 0.05,
    pathLength: 0.03,
    numQueryParams: 0.07,
    hasSuspiciousChars: 0.02,
    subdomains: 0.05,
  };

  // Compute the weighted sum
  let score = 0;
  for (const feature in features) {
    score += coefficients[feature] * features[feature];
  }

  // Logistic function to map the score to a probability
  const probability = 1 / (1 + Math.exp(-score));

  // Malicious threshold (e.g., 0.5 indicates 50% chance)
  const threshold = 0.5;

  return probability >= threshold;
}
```

The “**checkIfMalicious(url)**” function that detect malicious URL using machine learning logistic regression model. The “feature” extracting the feature of URL for logistic regression, the logistic regression model uses the hardcoded coefficients and threshold. The “score” feature compute the weighted of the features sum. The “probability” logistic function use to map the score to a probability. The “threshold” function check for malicious threshold e.g. 0.5 indicates 50% chance of being malicious.

Figure 19 URL Detecting.

Connect server:

```
// Get the URL input value
var urlInput = document.getElementById('url-input').value;

// Create a POST request to the Flask server
fetch('http://127.0.0.1:8080/detect', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json'
  },
  body: JSON.stringify({ url: urlInput }) // Send the URL in JSON format
})
.then(response => response.json()) // Parse the JSON response
.then(data => {
  // Display the result in the "result" div
  var resultDiv = document.getElementById('result');
  if (data.is_suspicious) {
    resultDiv.innerHTML = `<span style="color: red;">Malicious URL: ${data.url}</span>`;
  } else {
    resultDiv.innerHTML = `<span style="color: green;">Benign URL: ${data.url}</span>`;
  }
})
.catch(error => {
  console.error('Error:', error); // Handle errors
});
```

The above image shows a POST request to the flask server, to send URL data in Json and parse the Json response from the server.

Figure 20 connect server.

Index.html:

```
<html lang="en">
<head>
  <title>SAFE SCANNER WEB APP</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
  <link href="https://fonts.googleapis.com/css?family=Montserrat" rel="stylesheet" type="text/css">
  <link href="https://fonts.googleapis.com/css?family=Lato" rel="stylesheet" type="text/css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
  <link rel="stylesheet" href="styles.css">
</head>
<body id="myPage" data-spy="scroll" data-target=".navbar" data-offset="60">

<nav class="navbar navbar-default navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-target="#myNavbar">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
    </div>
    <div class="collapse navbar-collapse" id="myNavbar">
      <ul class="nav navbar-nav navbar-fixed-top">
        <a class="navbar-brand" href="#myPage">
          
        </a>
        <li><a href="/index.html">HOME</a></li>
        <li><a href="#about">URL SCANNER </a></li>
        <li><a href="/qrScanner.html">QR CODE SCANNER</a></li>
        <li><a href="#services">SERVICES</a></li>
        <li><a href="#contact">CONTACT</a></li>
      </ul>
    </div>
  </div>
</nav>
```

Figure 21 Home page.

styles.css

```
/* style.css */
body {
  font: 400 15px Lato, sans-serif;
  line-height: 1.8;
  color: #818181;
}
h2 {
  font-size: 24px;
  text-transform: uppercase;
  color: #303030;
  font-weight: 600;
  margin-bottom: 30px;
}
h4 {
  font-size: 19px;
  line-height: 1.375em;
  color: #303030;
  font-weight: 400;
  margin-bottom: 30px;
}
.jumbotron {
  background-color: #00A18C;
  color: #fff;
  font-family: Montserrat, sans-serif;
}
```

Figure 22 style file.

Index.html is the HOME page of the web application, it also contains the navigation bar, bootstraps are used to create navigation bar, and CSS is used for styling of the web page.

## 2.4. Graphical User Interface (GUI)

Logo:



Figure 23 project Logo.

The above image is the Logo of the Safe Scanner web application.

Home page and navigation bar:

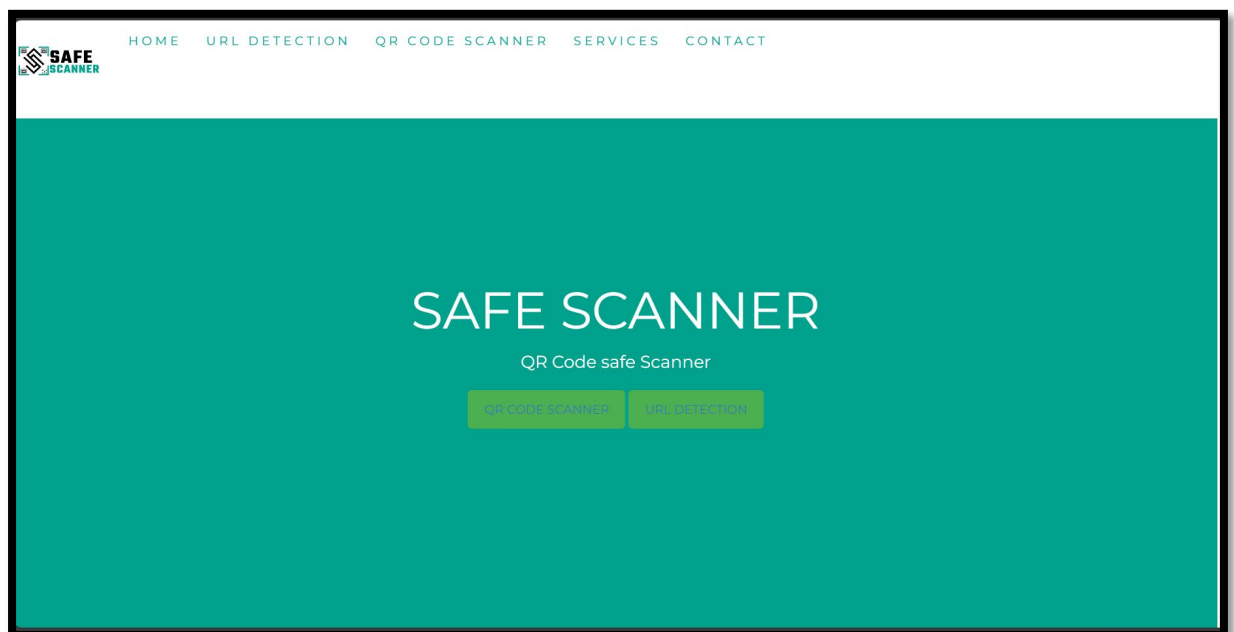


Figure 24 Home page and navigation bar.

The above image is the Home page and navigation bar of Safe Scanner web application, the navigation bar present user with various links, to access the home page click in Logo or the

“HOME”, to access to contact information click in “CONTACT”, to navigate to QR code scanner page click in “QR CODE SCANNER”, to navigate to URL scanner page click in “URL SCANNER”.

The make it easy for user to scan QR code home page also contains two buttons, “QR CODE SCANNER” and “URL SCANNER”.

QR code scanner page:

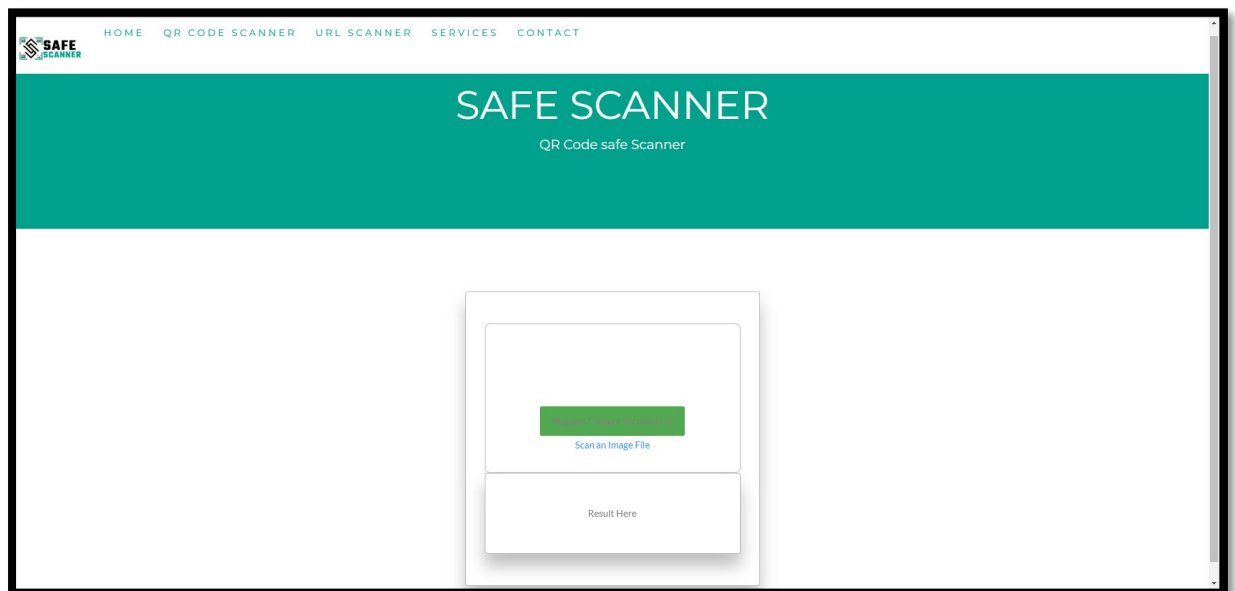


Figure 25 QR code scanner page.

The above image is the QR code scanner page, user are able to scan QR code by scanning QR code by real time camera or by uploading QR code image to scan and the result will be displayed. Figure 23.

Camera access.

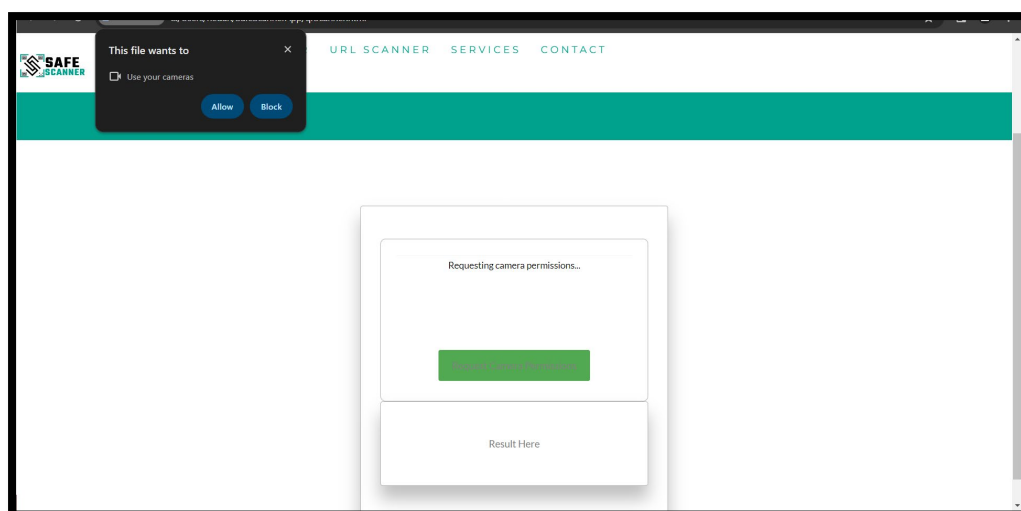


Figure 26 Camera access.

The above image shows to access device camera the application request for camera access. Where user can allow and block the access to camera. Figure 24

Scan code:

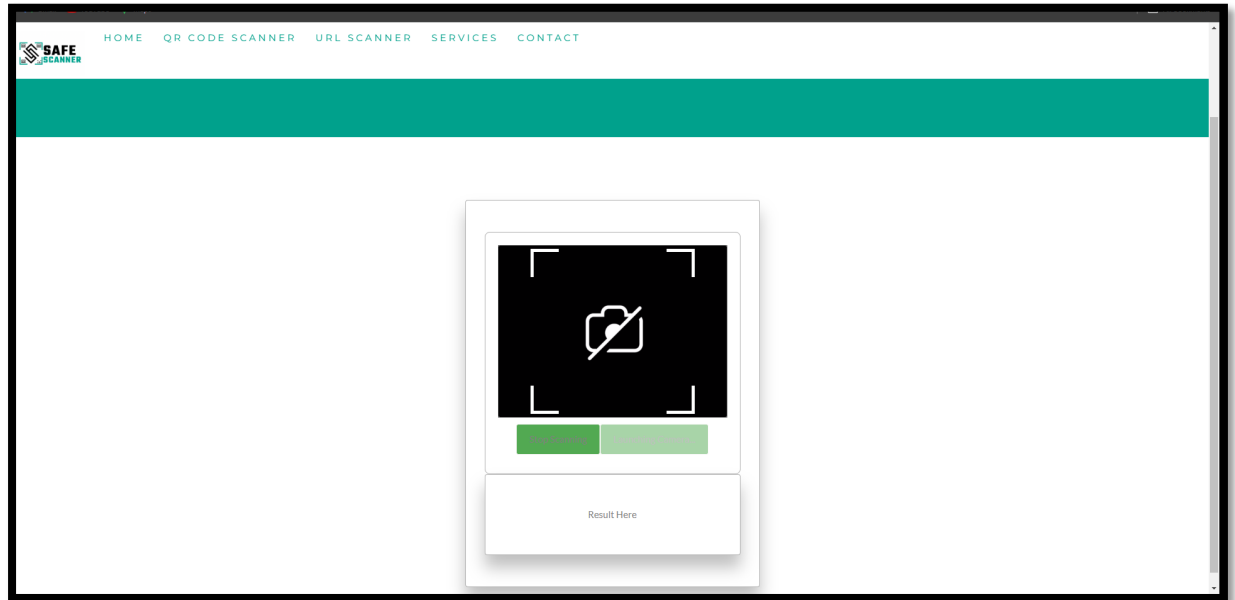


Figure 27 Scan Code.

The above image shows the application ready to scan, user open camera and direct the QR code to scan, Figure 25

Malicious QR Scan:

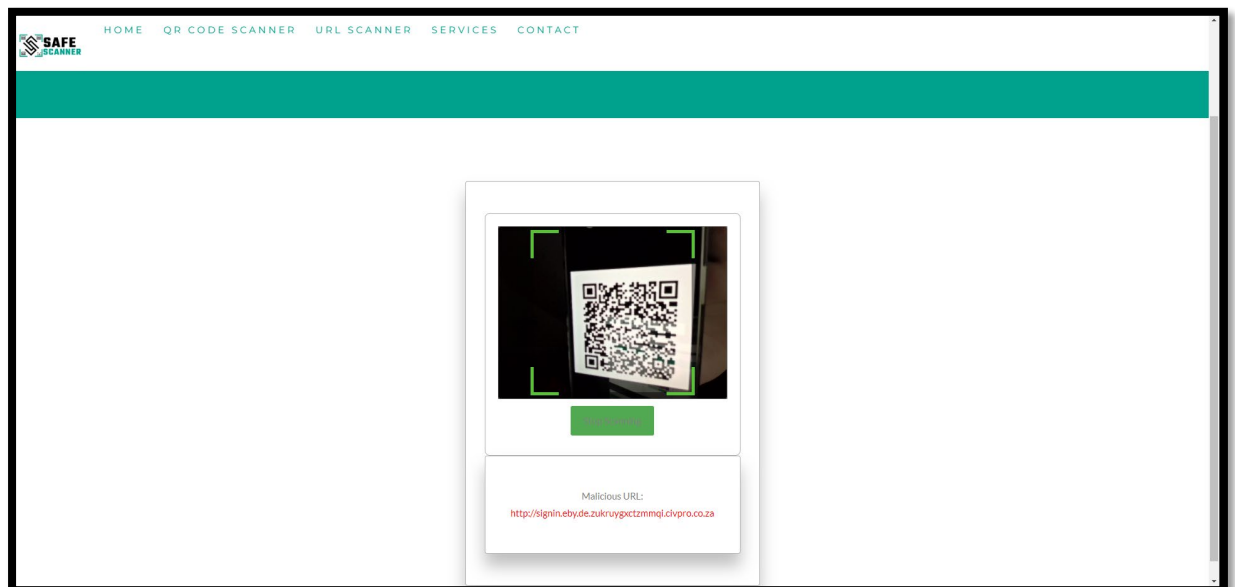


Figure 28 Malicious QR Scan.

The above image shows that a Malicious QR code have been scanned and the result have been displayed “Malicious URL: url”. As this is malicious URL the user cannot access the URL. Figure 26

Benign QR code:

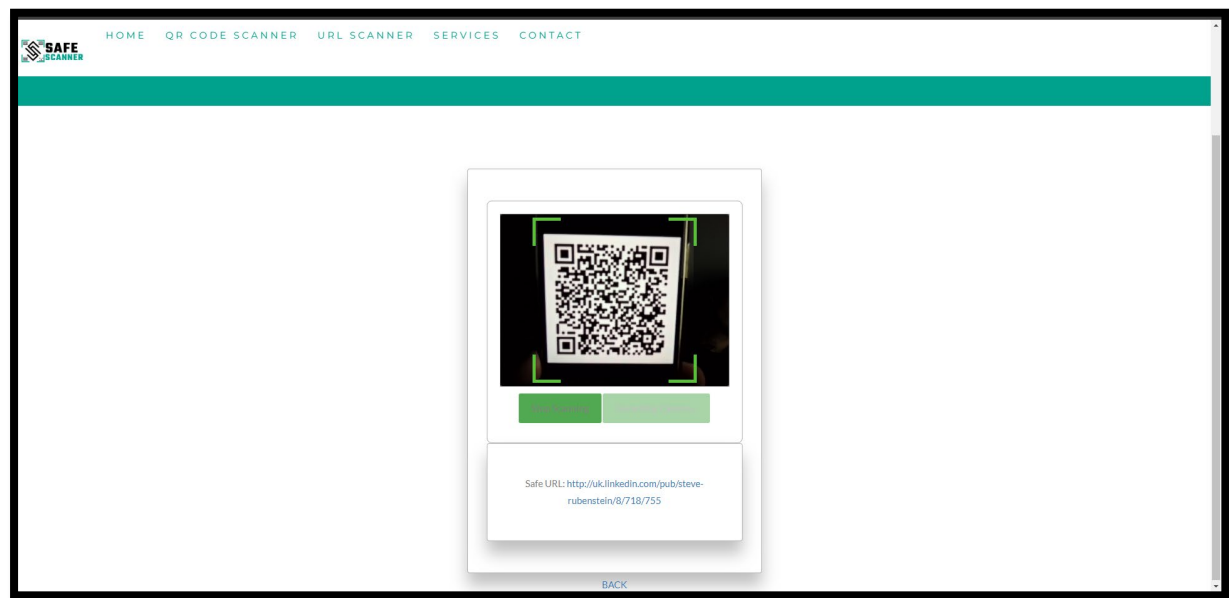


Figure 29 Benign QR code.

The above image shows that a benign QR code have been scanned for information and the result has been displayed “Safe URL: url”. As this is a safe URL the user is able to access URL by click on URL. Figure 27

Image Scan:

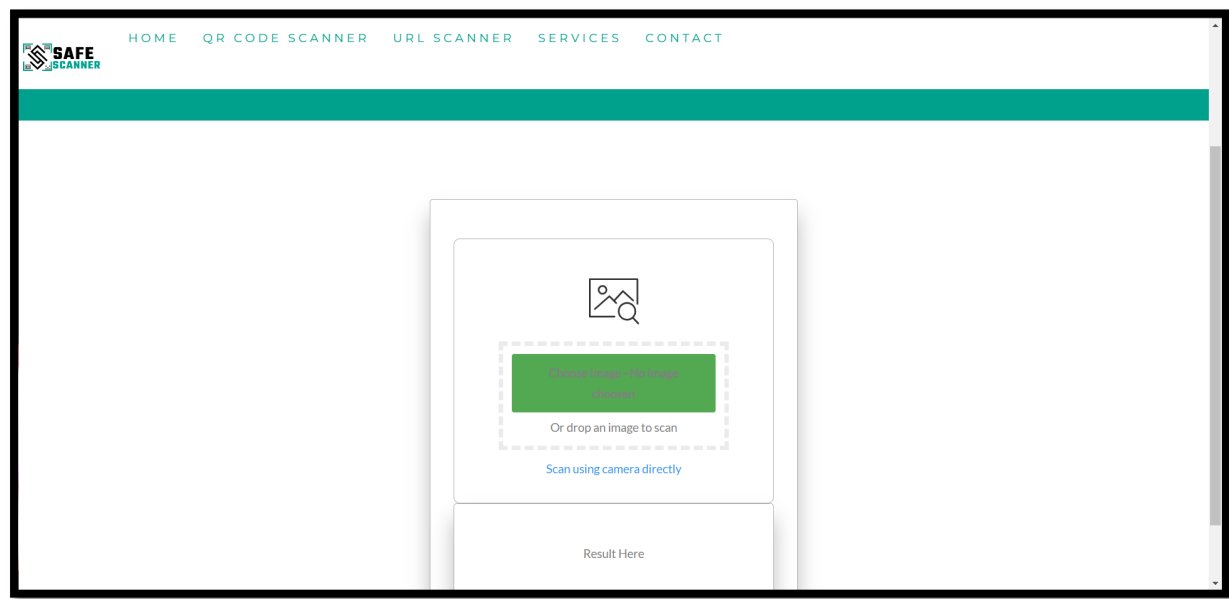


Figure 30 Image Scan.

The above image shows the application are able to scan an image as well, the user can choose QR code image to scan for detection. The application can access user file to upload image or user can drag and drop the image to scan QR code for information. Figure 28

Benign QR code image scanned:

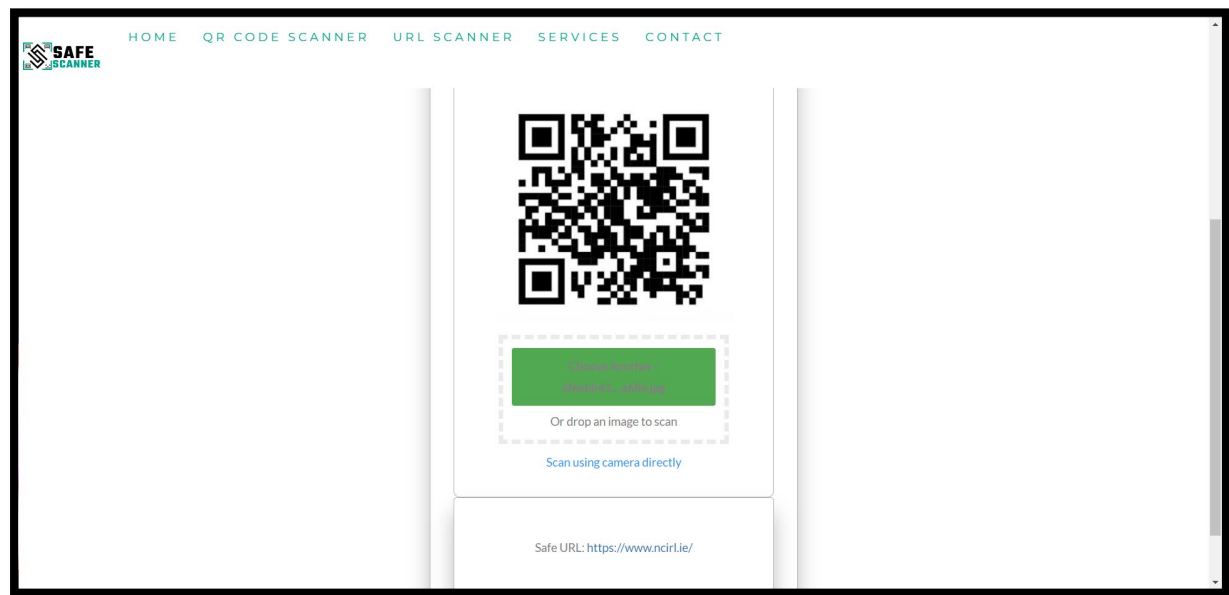


Figure 31 Benign QR code image.

The above image shows that a benign image has been darg for scanning and the result have been displayed to user at result section. displayed result “Safe URL: url”. As this is a safe URL the user can access URL by click on URL. Figure 29

Malicious QR code image:

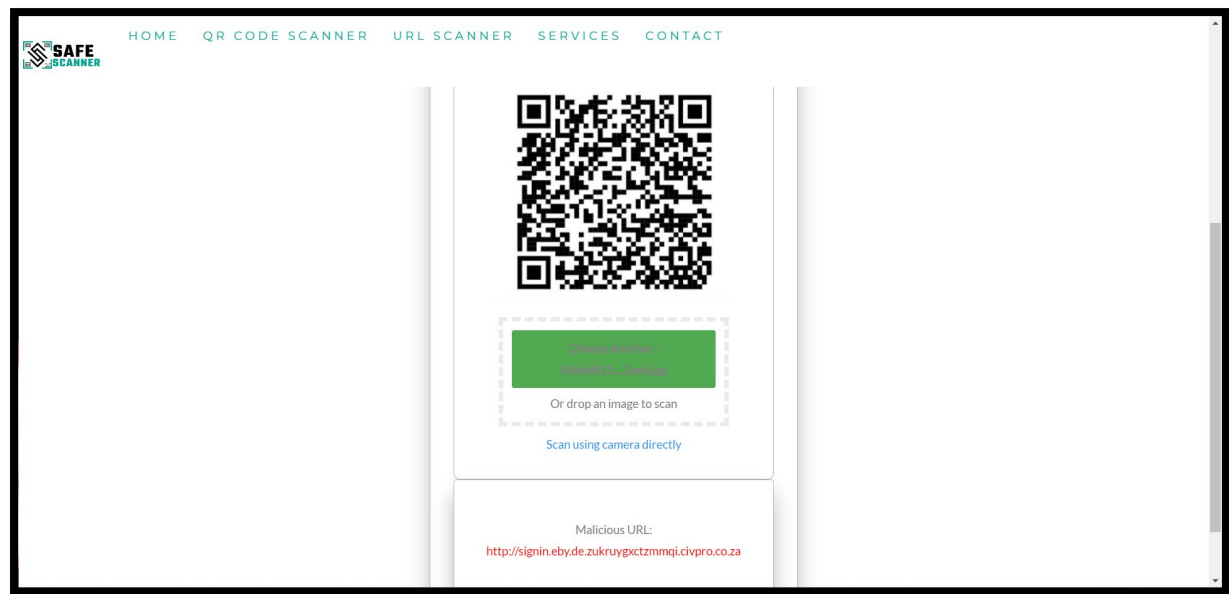


Figure 32 Malicious QR code image.

The above image shows that a Malicious QR code image has been drag for scanning and the result have been displayed “Malicious URL: url”. As this is malicious URL the user cannot access the URL. Figure 30

URL Scanner page:

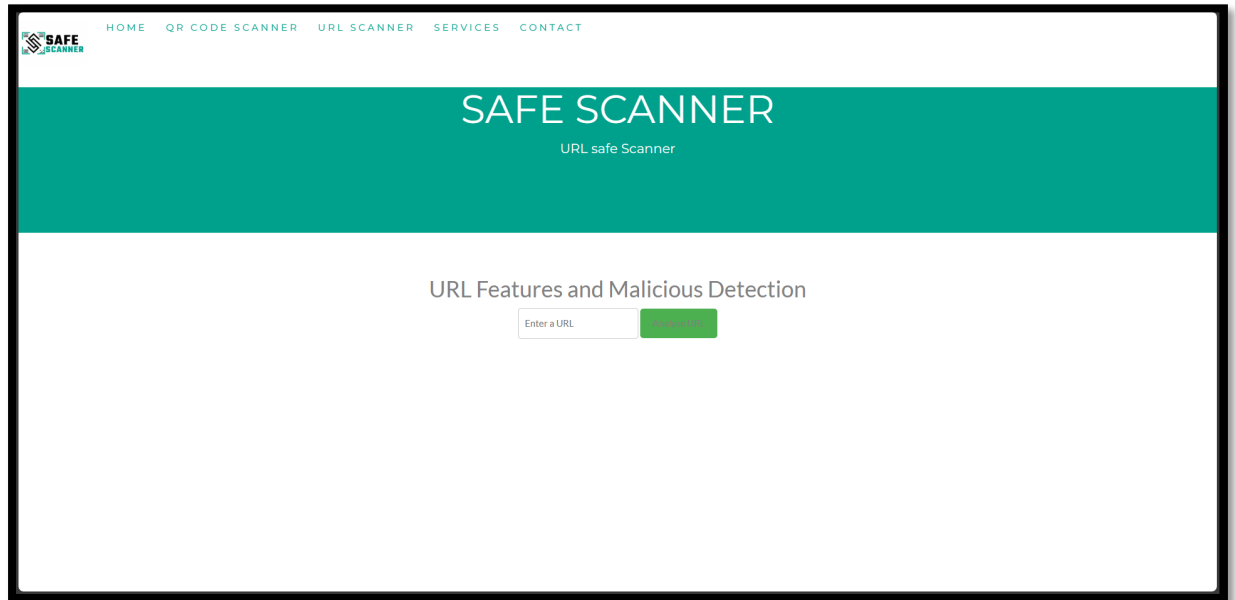


Figure 33 URL Scanner page.

The above image is the URL scanner page, user are able to enter a URL in “Enter URL” section and press detect for detecting the URL if its benign or malicious and extract features of URL such as scheme, authority, path, and query. After clicking the Detect the result will be displayed.

Figure 31

Benign URL detection:



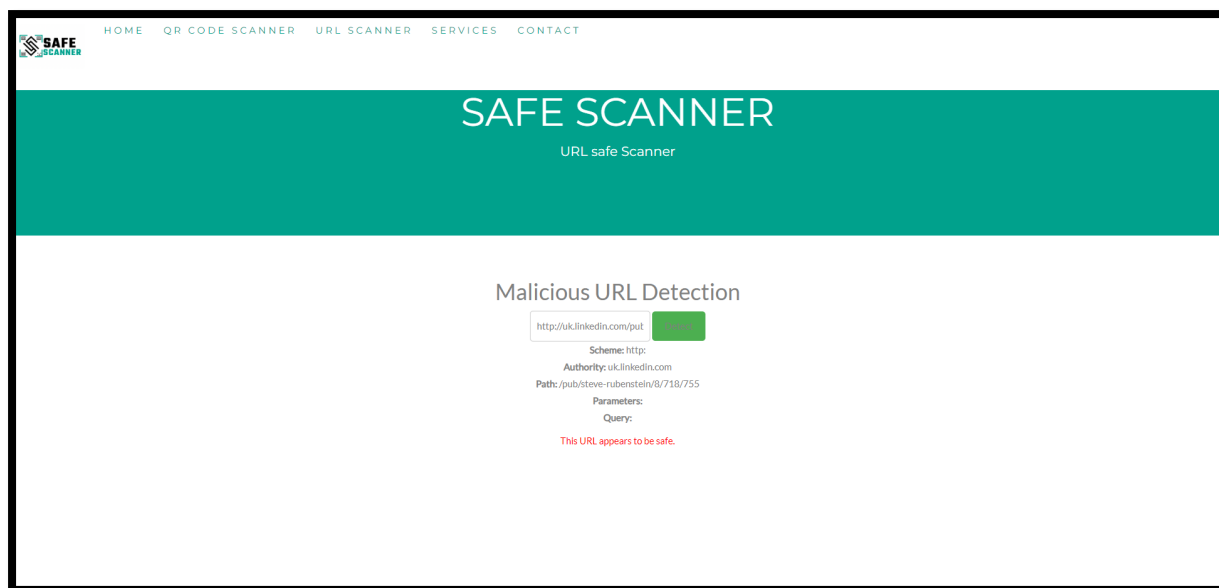


Figure 34 Benign URL detection.

The above image shows that a benign URL has been entered and the result has been displayed, the URL feature and "The URL appears to be Safe". Figure 32

Malicious URL detection:

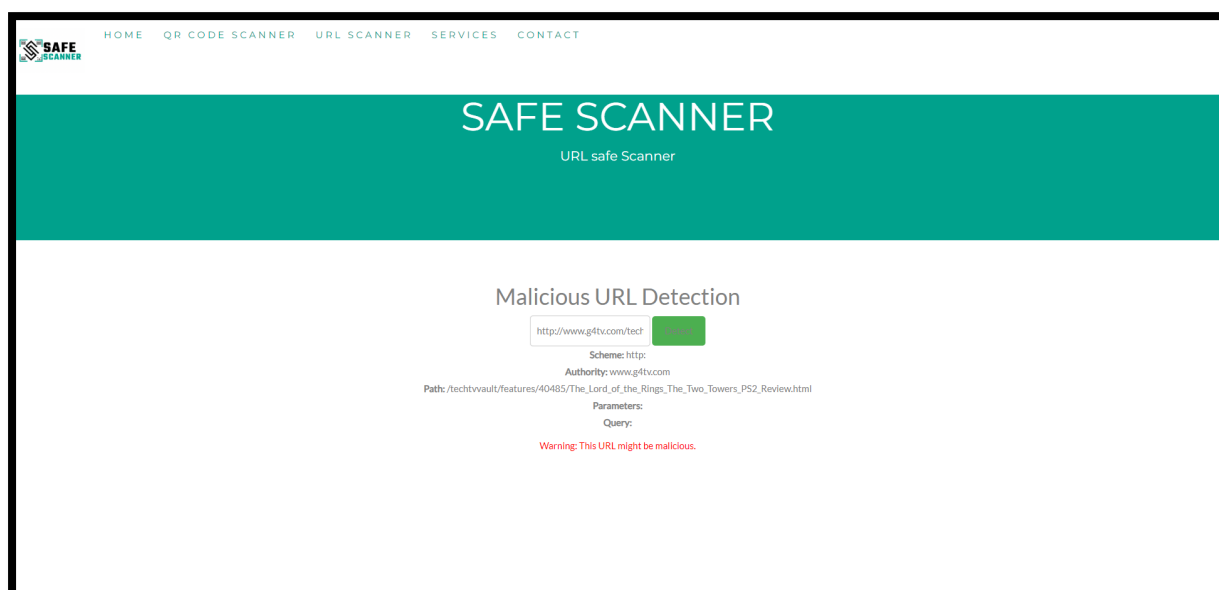


Figure 35 Malicious URL Detection.

The above image shows that a malicious URL has been entered and the result has been displayed "Warning the URL might be malicious" and the URL features have been displayed. Figure 33

Contact page:

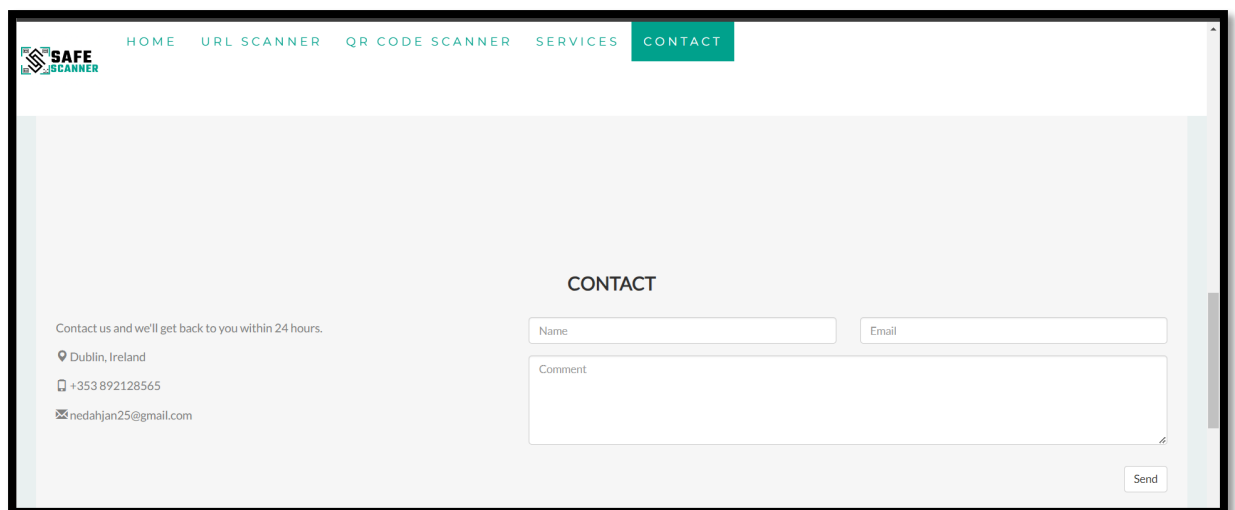


Figure 36 Contact page.

The above image shows the contact page of the application for user feedback and interaction.  
Figure 34

## 2.5. Flow Diagram

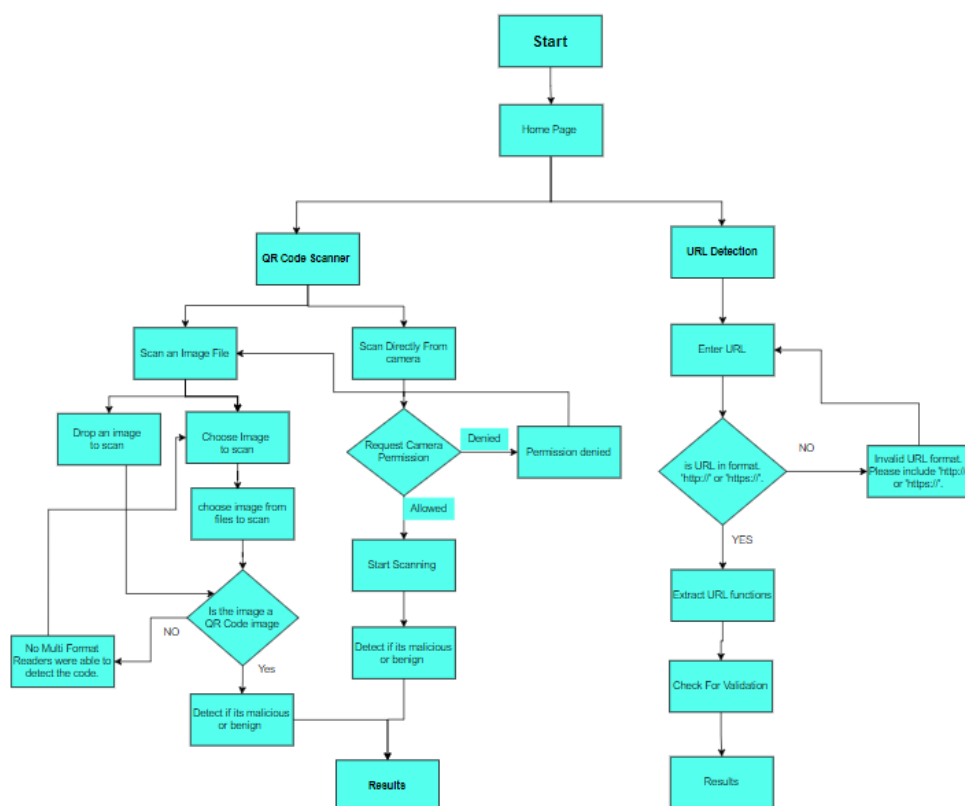


Figure 37 actively Flow Diagram.

## 2.6. Testing

### System Testing:

System testing validated the end-to-end functionality of the project. Key focus areas included QR code scanning, content handling, user interface behaviour, and error handling. Security and performance tests were also part of system testing.

### Integration Testing:

Integration testing ensured proper interaction between system components. This included testing the QR code scanner, content type handler, and output display to ensure seamless integration.

### Class Testing:

Class testing focused on individual classes to ensure their methods and attributes worked correctly. This helped identify defects at a more granular level before they could affect the entire system.

### Unit Testing:

Unit testing validated individual functions and methods. Automated unit tests were used to ensure code correctness and robustness, with a focus on boundary cases and edge cases.

- Scanning QR Codes: Test with valid QR codes to ensure proper recognition and processing.
- Content Type Determination: Check the handling of different content types (URLs, plain text).
- URL Feature Extraction: Validate the extraction and classification of URL features.
- Display Outputs: Ensure correct rendering of messages, links, and styles.
- Error Handling: Test various error scenarios, such as invalid QR codes, scanning failures, and network issues.

### Testing tools:

#### Selenium IDE:

Selenium is a powerful framework for automated testing of web applications. It supports a variety of browsers and provides a robust set of tools for simulating user interactions and verifying application behaviour. Selenium Integrated Development Environment (IDE) is an integrated development environment specifically designed for creating Selenium test scripts in a simplified, user-friendly manner. (selenium-framework, 2023)

**unittest:** Python testing framework, the unittest module provides a rich set of tools for constructing and running tests. (unittest\_framework, 2024)

**Pytest:** is a simple testing framework written in Python and for Python. it provides a cleaner and shorter way of writing tests in Python. For instance, validating a code output is as straightforward as calling an assert keyword. (Unit Testing in Python, 2022)

### System test:

Test ID	Function	Description	Expected Results	Actual Results	Test Outcome
T1	Navbar Links test	1. Access Home page. 2. Click on links in navbar. 3. Verify Navigates to different pages.	User able to navigate through the navbar links	The User was able to navigate through the navbar links	Pass
T2	Scan QR code	1. Click on QR Code Scanner. 2. Click Start Scanning. 3. Give permission to open camera. 4. Place QR code to camera. 5. Scan QR code.	Result of QR Code is displayed. Benign/Malicious URL: link of URL	The user was able to scan QR code successfully, the result displayed: Benign URL: link	Pass
T3	Detect URL	1 access Home page. 2. click on URL Scanner. 3. Enter URL. 4. Click Detect URL.	URL detected. Result displayed.	URL was detected the Result displayed.	Pass
T4	URL Feature Extraction	1.Enter URL. 2. Click Detect URL	The URL extracted feature are displayed.	The URL extracted feature are displayed.	Pass
T5	Scanning QR code from image	1. Click on QR Code Scanner. 2. Click Scan an Image File. 3. Choose Image or drop an image to scan.	Result displayed in Result section	Result was displayed in Result section	Pass
T6	Error Handling	Entered URL in wrong format e.g. not including http:// or https://	Invalid URL format. Please include 'http://' or 'https://'.	Invalid URL format. Please include 'http://' or 'https://'.	Pass

Table 1 Testing.

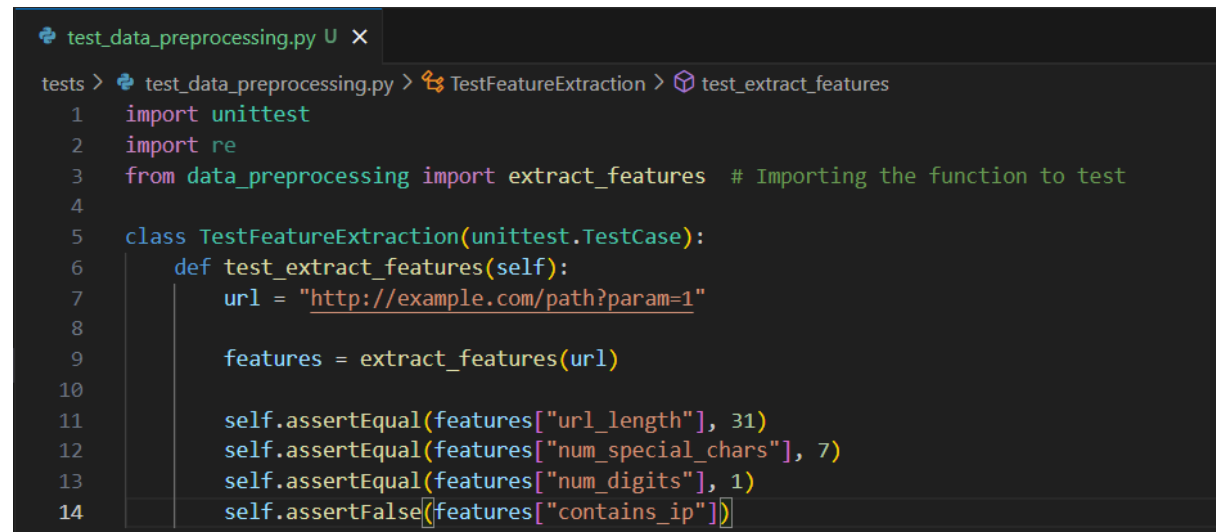
Manual testing result: the grid detailing the result of manual testing in the browser. Table 1

### Unit Testing:

Unit testing validated individual functions and methods. Automated unit tests were used to ensure code correctness and robustness, with a focus on boundary cases and edge cases. I've

used unit test in my project to cover the critical part of the project, also to ensure that the project work as expected. By implementing these tests want to maintain a high level of confidence in the system functionality and quickly identify issues when changes are made.

### Data preprocessing test:

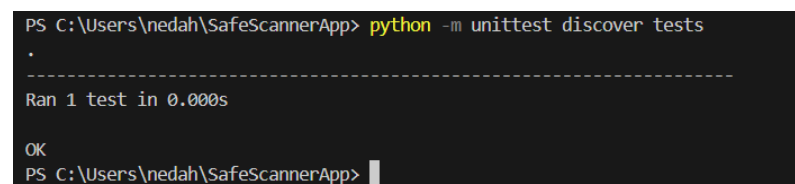


```
test_data_preprocessing.py U x
tests > test_data_preprocessing.py > TestFeatureExtraction > test_extract_features
1 import unittest
2 import re
3 from data_preprocessing import extract_features # Importing the function to test
4
5 class TestFeatureExtraction(unittest.TestCase):
6     def test_extract_features(self):
7         url = "http://example.com/path?param=1"
8
9         features = extract_features(url)
10
11         self.assertEqual(features["url_length"], 31)
12         self.assertEqual(features["num_special_chars"], 7)
13         self.assertEqual(features["num_digits"], 1)
14         self.assertFalse(features["contains_ip"])]
```

Figure 38 Data Preprocessing test.

The “test\_data\_preprocessing.py” file have been crated to test for the data preprocessing, such as to test the feature extracting and label mapping. Figure 35

### The unittest result:



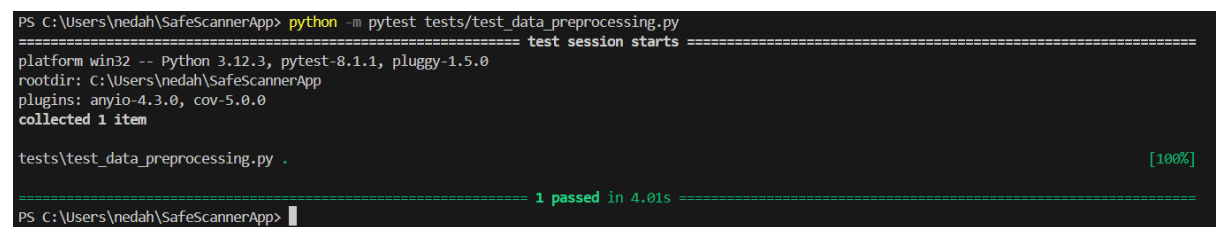
```
PS C:\Users\nedah\SafeScannerApp> python -m unittest discover tests
.
-----
Ran 1 test in 0.000s

OK
PS C:\Users\nedah\SafeScannerApp>
```

Figure 39 Data Preprocessing Test Result.

The test label mapping verifies that the label mapping converts textual label to numeric values. Figure 36

### The pytest result:



```
PS C:\Users\nedah\SafeScannerApp> python -m pytest tests/test_data_preprocessing.py
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.1.1, pluggy-1.5.0
rootdir: C:\Users\nedah\SafeScannerApp
plugins: anyio-4.3.0, cov-5.0.0
collected 1 item

tests\test_data_preprocessing.py .                                     [100%]

===== 1 passed in 4.01s =====
PS C:\Users\nedah\SafeScannerApp>
```

Figure 40 Data Preprocessing Test Result.

## Feature Extraction Testing:

```
test_train_model.py > TestTrainModel > test_train_model
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import unittest
import re

class TestTrainModel(unittest.TestCase):
    def test_train_model(self):
        # Sample data for testing
        data = {
            "url_length": [10, 20, 30],
            "num_special_chars": [1, 2, 3],
            "num_digits": [0, 1, 2],
            "contains_ip": [False, False, True],
            "label": [0, 1, 2]
        }

        df = pd.DataFrame(data)

        # Split into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(
            df.drop("label", axis=1),
            df["label"],
            test_size=0.3,
            random_state=42
        )
```

Figure 42 Feature Extraction Testing.

```
test_train_model.py > TestTrainModel > test_train_model
class TestTrainModel(unittest.TestCase):
    def test_train_model(self):
        )

        # Standardize the features
        scaler = StandardScaler()
        X_train_scaled = scaler.fit_transform(X_train)
        X_test_scaled = scaler.transform(X_test)

        # Train the model
        model = RandomForestClassifier(n_estimators=10, random_state=42)
        model.fit(X_train_scaled, y_train)

        # Test if the model predicts without errors
        predictions = model.predict(X_test_scaled)

        self.assertEqual(len(predictions), len(y_test))
```

Figure 41 Train The model.

The “RandomForestClassifier” function used to train the model for detecting malicious URL. The TestTrainModel testing function used to test for the model training. Figure 38

The “test\_train\_model.py” test verifies that the expected features are created from a given URL using the feature extraction function. Figure 39

### The unittest result:

```
PS C:\Users\nedah\SafeScannerApp> python -m unittest discover tests
..
-----
Ran 2 tests in 0.008s

OK
PS C:\Users\nedah\SafeScannerApp> █
```

Figure 43 Test train model testing result.

Result of “test\_train\_model.py” test, Successful. Figure 40

### The pytest result:

```

PS C:\Users\nedah\SafeScannerApp> python -m pytest tests/test_train_model.py
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.1.1, pluggy-1.5.0
rootdir: C:\Users\nedah\SafeScannerApp
plugins: anyio-4.3.0, cov-5.0.0
collected 1 item

tests\test_train_model.py . [100%]

===== 1 passed in 0.78s =====
PS C:\Users\nedah\SafeScannerApp>

```

Figure 44 test train model testing result.

## Flask endpoint testing:

```

t_flask_api.py 1, U X
> test_flask_api.py > ...
from flask import Flask, request, jsonify
import unittest
import json

app = Flask(__name__)

@app.route("/detect", methods=["POST"])
def detect_url():
    data = request.json
    url = data.get("url", "")

    # Dummy response for unit testing
    response = {
        "url": url,
        "is_suspicious": False,
        "message": "Benign URL"
    }

    return jsonify(response)

```

Figure 46 flask endpoint testing.

```

    return jsonify(response)

class TestFlaskAPI(unittest.TestCase):
    def setup(self):
        self.app = app.test_client()
        self.app.testing = True

    def test_detect_url(self):
        # Send a sample POST request to the Flask endpoint
        response = self.app.post(
            "/detect",
            data=json.dumps({"url": "http://example.com"}),
            content_type="application/json"
        )

        # Test if the response is valid and contains the expected fields
        data = json.loads(response.data)

        self.assertEqual(data["url"], "http://example.com")
        self.assertFalse(data["is_suspicious"])
        self.assertEqual(data["message"], "Benign URL")

```

Figure 45 flask endpoint testing.

The “test\_flask.\_api.py” test verifies that the flask-based server correctly accept requests, access the input and return a valid result. Figure 42, Figure 43

## The unittest result:

```

PS C:\Users\nedah\SafeScannerApp> python -m unittest discover tests
...
-----
Ran 3 tests in 0.018s

OK
PS C:\Users\nedah\SafeScannerApp> 

```

Figure 47 flask end point testing result.

The result of “test\_flask.\_api.py” test using unittest tools testing has been Successful. The testing is done in 0.018s. Figure 44

## The pytest result:

```
PS C:\Users\nedah\SafeScannerApp> python -m pytest tests/test_flask_api.py
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.1.1, pluggy-1.5.0
rootdir: C:\Users\nedah\SafeScannerApp
plugins: anyio-4.3.0, cov-5.0.0
collected 1 item

tests\test_flask_api.py . [100%]

===== 1 passed in 0.12s =====
PS C:\Users\nedah\SafeScannerApp>
```

Figure 48 flask end point testing results.

The test\_flask\_api.py have been tested using pytest python testing tool, the test was successfully done in 0.12s result 100%. Figure 45

## The pytest result:

```
PS C:\Users\nedah\SafeScannerApp> python -m pytest
===== test session starts =====
platform win32 -- Python 3.12.3, pytest-8.1.1, pluggy-1.5.0
rootdir: C:\Users\nedah\SafeScannerApp
plugins: anyio-4.3.0, cov-5.0.0
collected 3 items

tests\test_data_preprocessing.py . [ 33%]
tests\test_flask_api.py . [ 66%]
tests\test_train_model.py . [100%]

===== 3 passed in 4.09s =====
PS C:\Users\nedah\SafeScannerApp>
```

Figure 49 test results.

the testing for all three classes has been done 100% successfully in 4.09s.

Result of the pytest testing. Figure 46

## Automated testing:

✓ click	id=html5-qrcode-button-file-selection	
✓ mouse out	id=html5-qrcode-button-file-selection	
X type	id=html5-qrcode-private-filescan-input	C:\fakepath\Scree03-07 002827.png

Figure 50 automated testing result.

Log	Reference
12. click on id=html5-qrcode-anchor-scan-type-change OK	
13. mouseOver on id=html5-qrcode-button-file-selection OK	
14. click on id=html5-qrcode-button-file-selection OK	
15. mouseOut on id=html5-qrcode-button-file-selection OK	
16. type on id=html5-qrcode-private-filescan-input with value C:\fakepath\Scree03-07 002827.png Failed: File uploading is only supported in Chrome at this time	
'testing Safe Scanner' ended with 1 error(s)	

Figure 51 automated testing.



The testing is done in selenium IED, the test fails for the image uploading for scanning, which have been resolved later. Figure 47, Figure 48

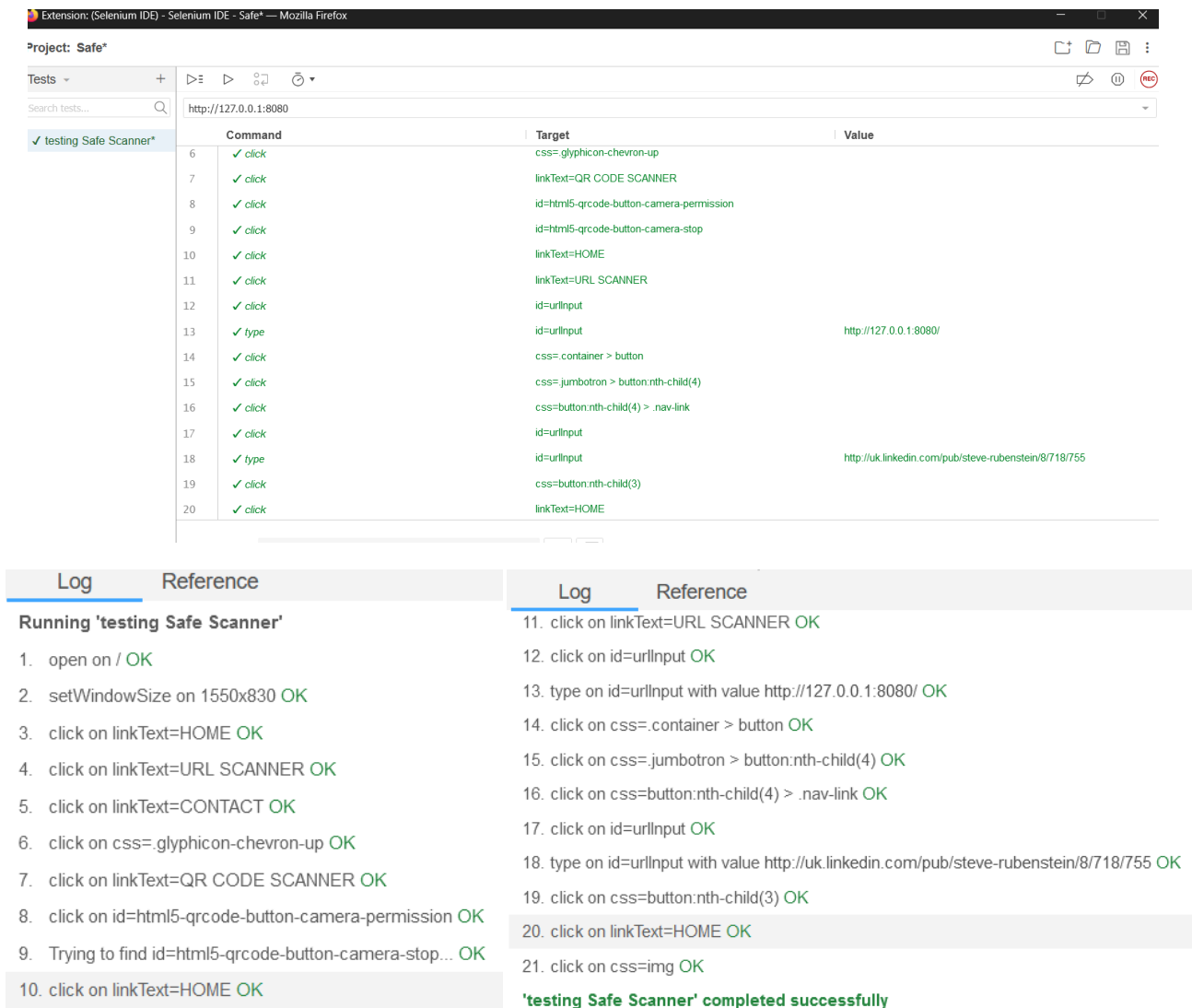
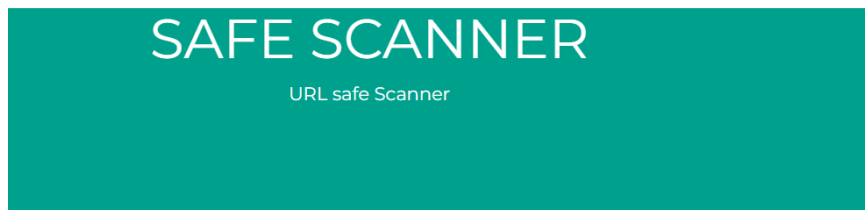


Figure 52 automated testing result.

Automated testing of the Safe Scanner application done using selenium IED. The selenium IED automated testing provide a user friendly to validate the behavior of the Safe Scanner application. The selenium testing creating automated test scenario to cover key use cases, to ensure that the application function works correctly and deliver seamless user experience. It has the ability to test repetitive to catch errors early in the development process. Figure 49

## Security Testing:



### Malicious URL Detection

Invalid URL format. Please include 'http://' or 'https://'.

Figure 53 cross-site scripting (XSS) in the web application.



### Malicious URL Detection

Invalid URL format. Please include 'http://' or 'https://'.

Figure 54 cross-site scripting (XSS) in the web application.

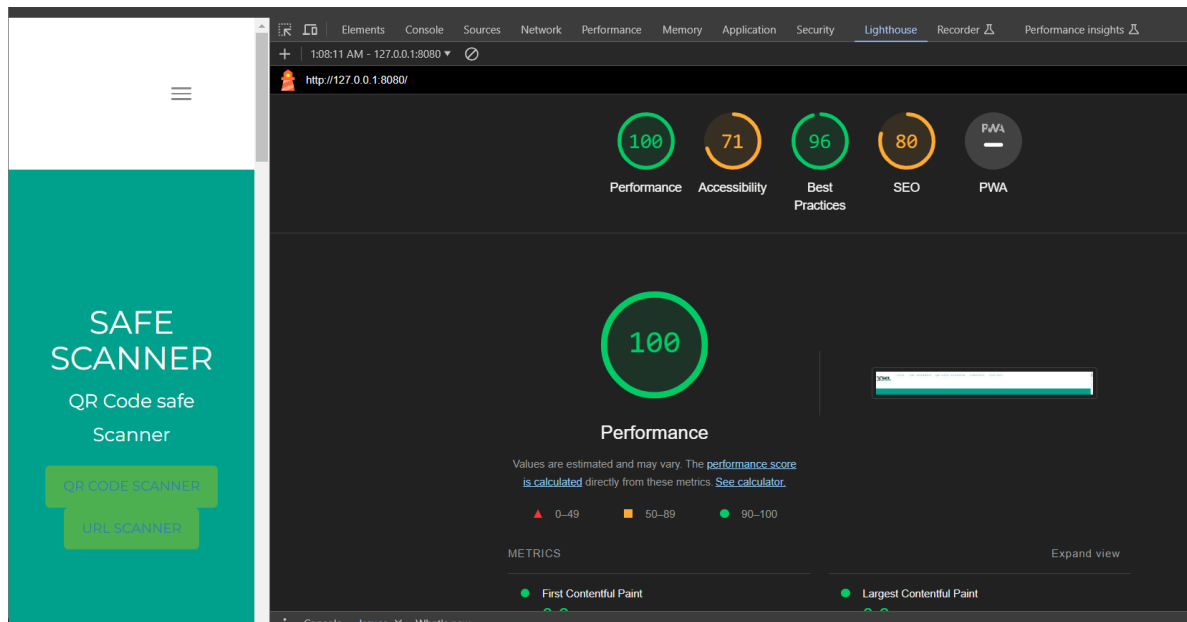


Figure 55 performance testing result.

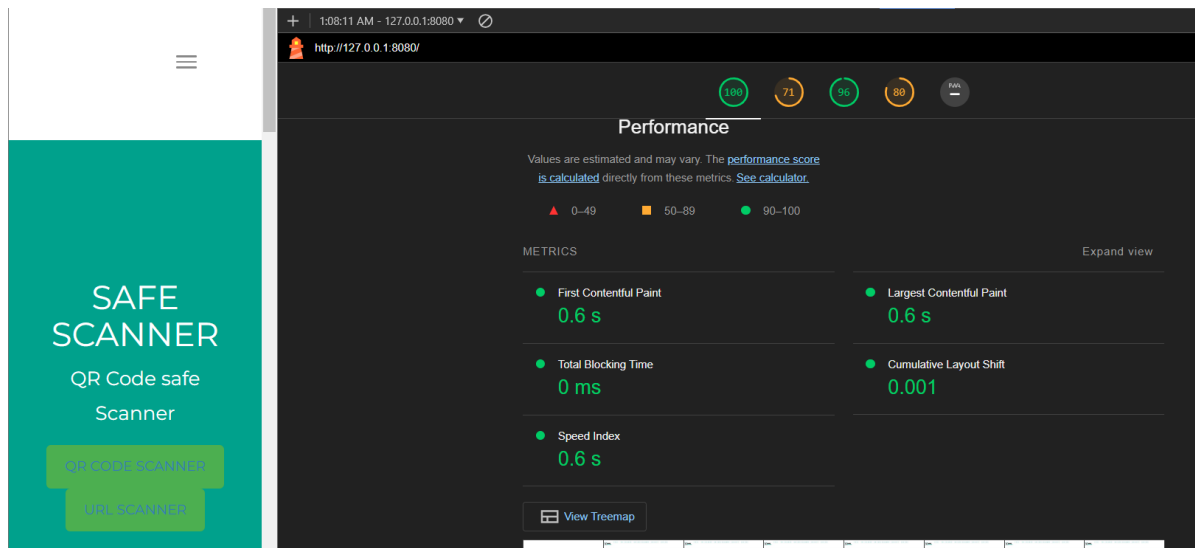


Figure 56 performance testing result.

The performance test has been done to measure the speed and responsiveness of the Safe scanner to ensure real time operation and minimal lag. To test the performance of Safe Scanner applications system, I've used Lighthouse an open-source chrome dev tool. To test for performance, accessibility, best practice, and SEO of the application. Figure 51

## 2.7. Evaluation

Evaluating the Safe Scanner system, it involves assessing its performance, functionality, correctness, and its scalability. The Safe Scanner web application was evaluated by running multiple tests. The application was tested with different variables successful information and tested with incorrect information.

	URL	Result
1	<a href="https://www.ncirl.ie">https://www.ncirl.ie</a>	Benign
2	<a href="https://stackoverflow.com">https://stackoverflow.com</a>	Benign
3	<a href="http://signin.eby.de.zukruygxtzmmqi.civpro.co.za">http://signin.eby.de.zukruygxtzmmqi.civpro.co.za</a>	Malicious
4	<a href="https://www.tutorialspoint.com/unittest_framework/index.htm">https://www.tutorialspoint.com/unittest_framework/index.htm</a>	Benign
5	<a href="http://www.marketingbyinternet.com/mo/e56508df639f6ce7d55c81ee3fcd5ba8/">http://www.marketingbyinternet.com/mo/e56508df639f6ce7d55c81ee3fcd5ba8/</a>	Malicious
6	<a href="http://friars.com/sports/m-baskbl/archive/prov-m-baskbl-2003.html">http://friars.com/sports/m-baskbl/archive/prov-m-baskbl-2003.html</a>	Benign
7	<a href="http://www.pn-wuppertal.de/links/2-linkseite/5-httpwwwkrebshilfede">http://www.pn-wuppertal.de/links/2-linkseite/5-httpwwwkrebshilfede</a>	Malicious
8	<a href="http://linkedin.com/pub/dir/elizabeth/scarborough">http://linkedin.com/pub/dir/elizabeth/scarborough</a>	Benign
9	<a href="http://fb.com.accounts.login.userid.343441.fbsbk.com/">http://fb.com.accounts.login.userid.343441.fbsbk.com/</a>	Malicious
10	<a href="http://www.vilagnomad.com/tables/payday-loans-direct-lenders-only.php">http://www.vilagnomad.com/tables/payday-loans-direct-lenders-only.php</a>	Malicious

Table 2 URL testing result.

The above URLs are taken from a benign and malicious data and have been tested in Safe Scanner application, the result have been stated above. Table 2

### Testing Result:

Test	Test 1	Test 2	Test 3	Test 4	Overall
Navbar Links test	success	success	success	success	All tests were successful
Scan QR code	success	success	success	success	All tests were successful
Detect URL	success	success	success	success	All tests were successful
URL Feature Extraction	success	success	success	success	All tests were successful
Scanning QR code from image	success	success	success	success	All tests were successful
Error Handling	success	success	Fail	success	Mix results, need to assess
Displayed Outcomes	success	success	success	success	All tests were successful

Table 3 over all testing results.

The application was evaluated by running various tested, unit test, integration test, system testing and automated tested. The test is resolved with different variables e.g. successful information and incorrect information to get best results. The overall results are successful, and I am happy with the results, only Test 3 the automated test in selenium has fail, which was solved later. Other than that, the testing result is successful. Table 3

### Performance check:

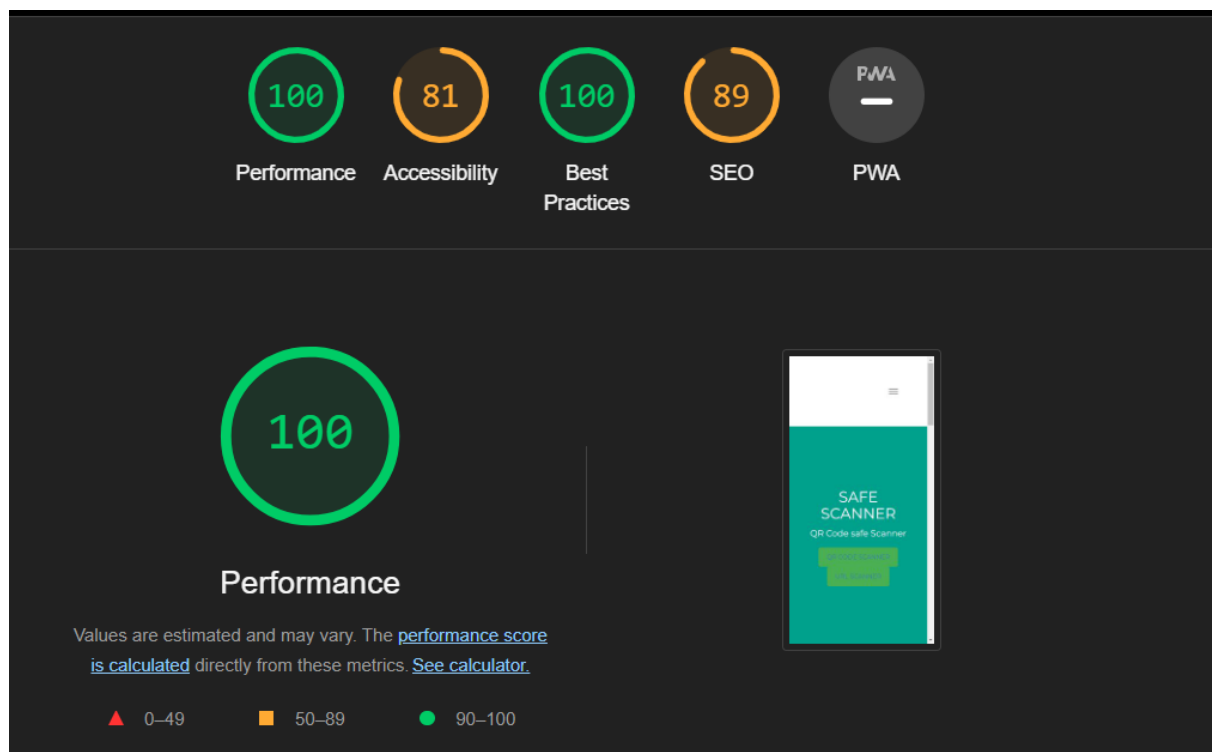


Figure 57 performance check result.

performance check score:

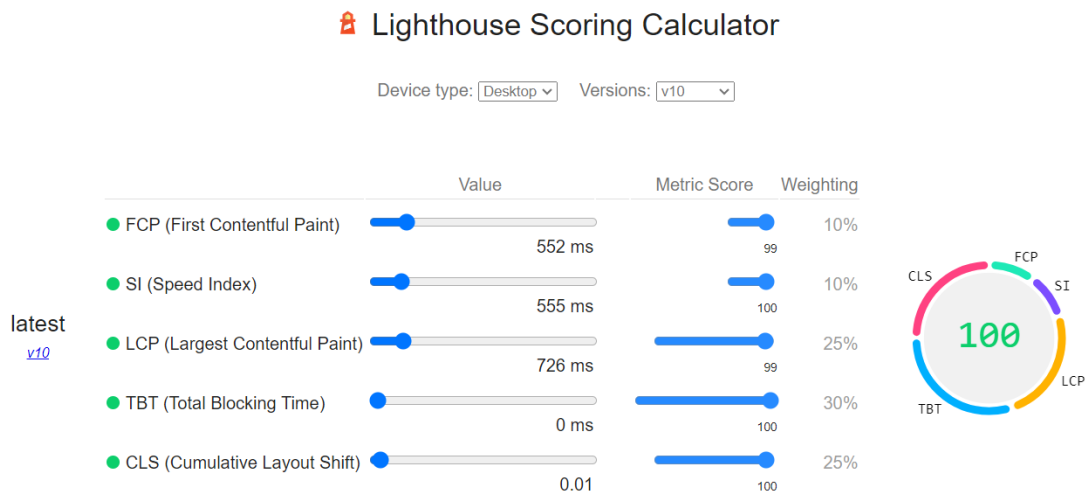


Figure 58 performance check score.

Lighthouse performance scoring.

To evaluate the performance of Safe Scanner applications system, I've used Lighthouse an open-source chrome dev tool. The performance evaluation of Safe Scanner shows that the application was responsive with scanning time of 0.6s. the performance level of Safe Scanner seemed sufficient for users. The Lighthouse performance scoring result assisted me the insight understanding into performance, accessibility, semantics, practices, and search engine optimisation. The result helped me improve and understand the area of improvement for further development of the Safe Scanner application. Figure 52, Figure 53

### 3.0 Conclusions

The QR code Safe Scanner project provides a utility tool to enhance the usability and security of the Safe Scanner application. There are several advantages to the Safe Scanner application, including better security and scalability. Some of the challenges facing the application include dependency on libraries that are not internal to the application. In general, the project promises to be in a good position to meet all user requirements for safe scanning of QR codes; however, to achieve this goal, continuous maintenance and development are required to realize the full potential that is inherent in the project.

**Advantages:**

- **Enhanced Security:** The system analyses malicious QRs, which might have dangerous threats to users. Increasing the security level is created.
- **URL Detection:** It tends to show high accuracy in distinguishing between malicious and benign URLs, and machine learning-based URL detection provides an effective defence against cybercriminal activities.

- **Real time Detection:** It allows users to use real-time QR code scanning, which helps to distinguish between innocuous and dangerous URLs and take the necessary measures.
- **Scalability:** Scalability is designed to satisfy user demand with minimum performance degradation for the system during modest workloads.
- **Modular design:** The project is broken into flexible components that are easy to maintain, update, or extend if required. Also, it may be tested and upgraded independently.
- **User-Friendly Interface:** to enhance usability The technology provides feedback to the user whether the scanned information is safe or hazardous.

#### Disadvantage:

- **Dependency on External Libraries:** A portion of the project relies on other frameworks and libraries, like the Flask framework and scikit-learn for machine learning. It adds dependencies that probably do require management and periodic updating.
- **Maintenance Overhead:** As the project grows in complexity and sophistication, the source code can become too hard to maintain and update, which demands constant effort to preserve efficiency and security.
- **User Dependence:** Through the real-time scanning of QR codes, it will have the ability to distinguish between both benign and dangerous URLs and take necessary action.

#### Strengths:

- **Comprehensive Functionality:** The application integrates the QR code scanning, and seamlessly displayed results. It provides for the user comprehensive solution.
- **Effective Detection:** The system effectively provides the needs for robust of QR code detection, a reliable tool for user to identify malicious threats.
- **User-Friendly Interface:** The frontend interface of the project makes the application accessible for all users, including those with no technical background in machine learning and cybersecurity.
- **Continuous Improvement:** The project facilitates ongoing improvement by structured process of development and the implementation of integration test and unit testing, it ensures that the project will remain effective and up to date.

## 4.0 Further Development or Research

Given additional time and resources the Safe Scanner project has few potential further development areas. These directions focus on improving the enhance system robustness, its functionality and security. Expanding its use cases and integration with other technologies.

The additional time and resources could be used to add new use case for the Safe Scanner project. Automatic scanning environment, the automatic environment will do analyses for Malicious QR code and URL, e.g. it will check if an email contain attachment. The Safe Scanner will analyse the QR code image if it contains URL, it will further navigate and detect if its malicious. Then it will block the email.

## 5.0 References

arxiv. (2019). *Malicious URL Detection using Machine Learning: A*, 37. Retrieved from arxiv.org:  
<https://arxiv.org/pdf/1701.07179>

*selenium-framework*. (2023, 02 06). Retrieved from browserstack.com:  
<https://www.browserstack.com/guide/selenium-framework>

*Unit Testing in Python*. (2022, 09 27). Retrieved from www.makeuseof.com:  
<https://www.makeuseof.com/python-testing-integration-unit-frameworks/>

*unittest\_framework*. (2024). Retrieved from www.tutorialspoint.com:  
[https://www.tutorialspoint.com/unittest\\_framework/unittest\\_framework\\_overview.htm](https://www.tutorialspoint.com/unittest_framework/unittest_framework_overview.htm)

arxiv. (2019). *Malicious URL Detection using Machine Learning: A*, 37. Retrieved from arxiv.org:  
<https://arxiv.org/pdf/1701.07179>

## 6.0 Appendices

Code Repository URL: <https://github.com/nedahjan25/SafeScanner>

### 6.1. Project Proposal



# National College of Ireland

## Project Proposal

### QR Code Security App

28/10/2023

BSCH

Cyber Security

2023/2024

Nedah Jan Safi

X20347946

X20347946@student.ncirl.ie

## 7.0 Objectives

The objective of the project is to upscale the security of QR code by simply scanning with the help of user-friendly app that can easily detect malicious QR codes. The primary goal is to raise the concern about the QR code risks, how to protect from phishing attacks, and to understand the differences between the harmful website and legit ones. In addition to this it will provide multifunction to detect QR code with a direct camera as well as to scan with help of an image and you can directly scan an URL. With following with this it ensure data privacy and help you to detect the potentially malicious QR Code as well as a dedication to always improving in light of new dangers.

## 8. Background

Why Choose to Undertake this Project?

The decision to undertake this project is driven by several compelling reasons:

- Primary thought to adopt this project as QR Code become very common to share and store information and can be easily used for phishing.
- This project is influenced by the need to enhance user security and ultimately guard people from being victims of cyber threats instigated by QR codes..

How to Meet the Objectives?

To ensure that this project meet the objective, we will employ various strategies:

- Cybersecurity-Centric Development: Develop an improved QR code scanner app with enhanced detection against harmful QR codes.
- Real-Time Assessment of threats: Conduct dynamic, real-time threat analysis to provide indications and warnings of emerging risks.
- User Education on Cyber Risks: Educate users about QR code and wider cybersecurity best practices.
- Phishing URLs Detection: Advanced techniques can be used to identify phishing sites and empower users to make secure choices.
- Feedback and reporting from user: implement user feedback on accuracy and threat reporting.
- Cross-Platform: Implement the application for different platforms to extend cybersecurity.
- Data Security and Privacy: Ensure data security and privacy in the light of cyber threats.
- Continuous Cybersecurity Improvement: Keeping system up to date against evolving threats.

## 9. State of the Art

There are similar applications and projects that already exist in this domain of QR code security. Most QR code scanner applications introduce some level of security in the features, such as link validation or data integrity checks. However, what differs this project and how work is different from others working on similar projects. It can be summarized as follows:

- Advanced Threat Detection: It performs advanced algorithms for real-time threat analysis beyond the base checks.
- User Education: In addition to security, it educates users on broader cybersecurity practices.
- Improvement from time to time: The project commits to continuous updates for leading-edge protection against emerging threats.

- Data Privacy: It focuses on the safety of the users' information and satisfies the requirements of legal and ethical norms.
- Cross-Platform Accessibility: It is a cross-platform application to help a larger audience enjoy all the advanced security features of the application.

## 10. Technical Approach

### 1. Development approach:

The QR code security project will be approached in a methodical and organized way to ensure that our objectives are met. The approach to development in the attainment of this QR code security project, identification of requirements, and how the project tasks are broken down is summarized below:

### 2. Requirements Identification:

- The intended functionality, database administration, and security will determine the technological requirements. And strives for ongoing development to address changing threats.
- Cybersecurity is a very important topic. As such, research is necessary in regard to the potential security concerns and dangers posed by QR codes and phishing. This will help in ascertaining how to keep the app safe.
- Our goal is not only to scan a QR code, but we also teach people. Need a strategic plan of the necessary information to include in the application by having users be more careful about online safety.
- The application should be supported on many devices; hence, there will be a conclusion on the platforms that it should be offered.

### 3. The breakdown of Requirements into Tasks and Development:

#### 1. Project Beginning:

- Gathering Requirements
- Initial project planning

#### 2. Design and Architecture:

- System architecture design
- User interface design
- Database structure design

#### 3. Development:

- Core application development.
- User education content creation
- Phishing site detection mechanism
- User feedback mechanism

#### 4. Cross-Platform Adaptation:

- Platform-specific adaptation and testing

#### 5. Security Implementation:

- Data security and privacy measures

#### 6. Testing and Quality Assurance:

- Testing, bug fixing, and quality assurance.

#### 7. Feedback Integration:

- Incorporating user feedback

#### 8. Deployment and Distribution:

- Launch on app stores.

Throughout the project, Regular targets shall be set up throughout the project to monitor progress and check if it is in line with the project duration and objectives.

## 11. Technical Details

### 1. Implementation of programming Language:

Programming Language: The choice of a programming language is critical. Popular programming languages that are widely used for developing mobile applications include:

- Python: Known for rich in readability and huge range of libraries.
- JavaScript: JavaScript is an object-based, dynamic language used by developers for creating dynamic web pages on almost all platforms.

Cross-Platform Frameworks: Alternatively, one might consider using cross-platform development frameworks like React Native or Flutter to create the app for both Android and iOS platforms. This approach can help in minimizing the time and expenses involved in development.

### 2. Principal Libraries:

The choice of libraries can significantly impact the speed of development, performance, and the overall reliability of the application. Required libraries for various components of the project may include.

- QR Code Scanning: The common libraries used for QR code scanning are ZBar and ZXing.
- Security Implementation: This can be custom code for specific algorithms implemented for security features and detection of advanced threats.
- User Interface: There are specific libraries and frameworks made for each platform, such as UIKit for iOS, Android UI for Android, and Tkinter or Kivy for cross-platform development.
- Data Handling and Encryption: OpenSSL along with other libraries implements such functionality for data encryption and secure data management.

### 3. Important Algorithms/Approaches Under Consideration:

- Pattern Recognition: For identification of QR code and determination of the integrity of the QR code.
- Checksum Verification: To guarantee that the data included in QR codes are accurate.
- Machine Learning and AI: For advanced threat detection, analyzing patterns, and identifying suspicious QR codes.
- Dynamic Analysis: Real-time analysis of QR codes to detect potential threats, including link validation and reputation checks.

- Phishing Site Detection: Employing techniques like URL parsing and cross-referencing with known phishing databases.
- Data Privacy and Encryption: Implementing encryption algorithms to secure user data and information.
- Continuous Improvement Pipeline: Designing a system for updating threat detection algorithms and educational content.
- User Reporting Mechanism: Implementing features for users to report suspicious QR codes.

## 12. Special Resources Required

Successful implementation of this project may necessitate the use of specialized resources, notably in the domains of development and security. These resources can include:

- Hardware and software for development.
- Tools and services that enhance security, such as threat intelligence.
- Tools for ensuring compliance with data privacy rules.
- Resources for creating educational content.
- Infrastructure for managing databases.
- Resources for receiving user input and implementing continual improvement.
- Evaluation setups.
- Proficiency in legal and compliance affairs.
- Proficiency in technical cybersecurity.

## 13. Project Plan

### **Project Plan for QR Code Security App:**

Phase 1: The plan and structuring.

1. Project Beginning.
  - Outline the requirements and goals of the project.
  - Evaluate the resources available
2. Gathering requirements and analysis
  - Conduct a investigation into the available security solutions for QR codes.
  - Determine tech needs.
3. Design and Architecture.
  - Create systems architecture and design the database.
  - Make a strategic plan in designing the user interface (UI).
  - Conclude the technical specs.
4. Technology and Resources.
  - Put in place development tools and integrated development environments (IDEs).
  - Obtain the correct hardware and equipment for testing.

Phase 2: Development.

5. App Development at the Core.
  - The major coding of the QR code scanning and warning mechanism should be done as soon as possible.
6. Implementation of Databases and Security Measures.

- Create a database with all the known harmful QR codes.
- Implement procedures to protect the privacy and security of data.

#### 7. Feature Development.

- Live identification capabilities for phishing sites.
- Design and implement tools for user input (feedback) and reporting.

#### Phase 3: Cross-Platform Adaptability and User Interface Design Development.

#### 8. User Interface (UI) development and design.

- Develop and design graphical elements for GUIs.
- Begin working on the user interface.

#### 9. Cross-Platform Adaptation.

- Porting the application to the Android and iOS platform.
- Carry out a test to identify and eliminate problems peculiar to a specific platform.

#### Phase 4: Quality Assurance and Testing.

#### 10. Initial Testing.

- Make the app work on both Android and iOS.
- Check for bugs that are special to the app.

#### 11. Addition of Feedback

- Incorporate user comments or feedback in the program.
- Perform extra testing on bugs if needed.

#### Phase 5: Implementation and record-keeping.

#### 13. The final testing phase and removal of the remaining software defects..

#### 14. Implementation Preparation.

- Get the app ready for release in app stores.

#### 15. Documentation and education for users.

- Develop user manuals and instructional materials.
- Create detailed documentation of the application.

#### 16. Submission and launch of the application.

- Upload the application to appropriate app stores.
- Launch the application for general availability to the wider audience.

#### Phase 6: Post-Deployment.

#### 17. Activities of monitoring and updating performed after the launch of a product or service.

- Constantly track user comments and performance of the app.
- Plan and perform the routine upgrades to ensure the safety and utility of the application.

## 14. Testing

To ensure the functionality, safety, and usability of the QR code security software, it is necessary to evaluate the system with real technical data and to involve end users. Some methods of approaching such assessments are outlined below:

**System Tests:**

- The test on QR Code Scanning and Detection will test the basic capabilities of the app by scanning a wide range of QR codes; this would include both benign codes and codes that are known to be malicious. Precise detection of malicious QR codes and warning should be made available to the system.
- Phishing Site Detection Test: The technique for detecting phishing sites can be validated in real time by scanning QR codes going to both authentic and phishing websites. Ensure that the system correctly recognizes phishing sites and issues due warnings.
- User Feedback System Test: The effectiveness of real-time phishing site detection can be tested by scanning the QR codes pointing to both genuine and fraudulent websites. It should be ensured that the system detects the phishing websites and alerts appropriately.

**Integration Tests:**

- Security Integration Test: Test the integration of security measures, including data encryption and privacy features. Confirm that user data is adequately protected.
- Cross-Platform Integration Test: If the app is developed for multiple platforms, ensure that it functions consistently across Android and iOS devices.
- User Feedback Integration Test: Test the integration of the user feedback system, making sure that user reports are correctly processed and contribute to database updates.

**Evaluation with End Users:**

- Usability Testing: Conduct usability testing sessions with end users to understand the overall usability of the application.
- User Feedback Analysis: Analyze the feedback collected from end users during the usability testing.
- User Education Evaluation: Ensure that users understand the content provided and can derive value from the educational component.
- Post-Launch Monitoring: Consistently monitor user comments and reviews on app stores once the app has been published.

**Evaluation with End Users:**

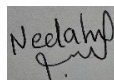
- Usability Testing: Conduct usability testing sessions with end users to understand the overall usability of the application.
- User Feedback Analysis: Analyze the feedback collected from end users during the usability testing.
- User Education Evaluation: Ensure that users understand the content provided and can derive value from the educational component.
- Post-Launch Monitoring: Consistently monitor user comments and reviews on app stores once the app has been published.

## 7.1. Reflective Journals

### 8.0 Supervision & Reflection Journal

<b>Student Name</b>	Nedah Jan Safi
<b>Student Number</b>	X20347946
<b>Course</b>	BSHCYB4
<b>Supervisor</b>	Vanessa Ayala-Rivera

### 9.0 Month: October

<b>What?</b>  In this month I have spent time on refining the project idea, thorough I have reviewed existing projects and Learned from the experiences of others has proven invaluable in shaping my approach. Produced a compelling Project Pitch Video and completed the project proposal.	
<b>So What?</b>  The dedicated research and ideation efforts significantly refined the project. now have a clearer understanding of the objectives and potential impact.  The exploration of existing projects provided valuable insights, helping with identify best practices and potential differentiators for the project.  The challenge that remains is to ensure to create a comprehensive roadmap for development.	
<b>Now What?</b>  What can you do to address outstanding challenges?  Break down the implementation into manageable milestones for better control.  Develop contingency plans for identified risks to minimize their impact on project timelines.  Set realistic deadlines for each phase of the project, considering potential challenges.  Seek feedback from supervisor.	
<b>Student Signature</b>	



## 10.0 Supervision & Reflection Journal

<b>Student Name</b>	Nedah Jan Safi
<b>Student Number</b>	X20347946
<b>Course</b>	BSHCYB4
<b>Supervisor</b>	Vanessa Ayala-Rivera

## 11.0 Month: November

### What?

In this month I have dedicated substantial time to in-depth research on the techniques relevant to the project.

Successfully crafted a detailed roadmap for the project's development.

Outlined key milestones, deliverables, and timelines, providing a structured plan for the project.

Conducted a comprehensive meeting with the project supervisor to discuss project objectives, expectations, and receive valuable guidance.

Clarified project scope, milestones, and sought feedback on the proposed approach.

### So What?

Achieved alignment with the project supervisor, ensuring our objectives are in sync with academic expectations.

Gained valuable insights from research, identifying the most effective techniques to be employed in project.

The development of roadmap provides a clear and organized plan.

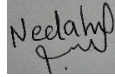
### Now What?

What can you do to address outstanding challenges?

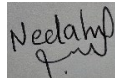
Identified potential challenges in integrating certain techniques, requiring careful consideration during the development phase.

With the roadmap in place, now must kick off the implementation phase in the coming month.

Implement contingency plans to ensure project continuity.

<b>Student Signature</b>	
--------------------------	---

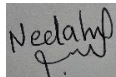
## 12.0 Month: December

<b>What?</b>  <p>In the past few weeks, I have delved into the technique for detecting QR code URLs as part of my project. This involved understanding various algorithms and methods for efficiently and accurately identifying URLs embedded within QR codes. I've explored the intricacies of image processing, pattern recognition, and the underlying principles that make QR code detection possible. The process involved a combination of literature review, experimentation, and engagement with relevant technologies.</p>	
<b>So What?</b>  <p>Understanding the QR code URL detection technique has been crucial for the foundation of my project. This knowledge has provided me with insights into the complexities and nuances of image processing and recognition. The so what aspect lies in recognizing the significance of this technique within the broader context of my project. I've been able to connect theoretical concepts to practical applications and foresee potential challenges that may arise during the implementation phase.</p>	
<b>Now What?</b>  <p>Begin planning the implementation phase, considering the intricacies of QR code URL detection. Considerations will include the selection of appropriate libraries, tools, and potential challenges that may arise during coding.</p>	
<b>Student Signature</b>	

## Supervision & Reflection Journal

<b>Student Name</b>	Nedah Jan Safi
<b>Student Number</b>	X20347946
<b>Course</b>	BSHCYB4
<b>Supervisor</b>	Vanessa Ayala-Rivera

**Month:**            **January**

<b>What?</b>  I have dedicated substantial time to designing the application's wireframe. The wireframe serves as the blueprint for the user interface, providing a visual representation of the application's structure and layout. It includes key elements such as navigation, buttons, and overall design aesthetics. The wireframe is crucial for ensuring a user-friendly and intuitive interface.	
<b>So What?</b>  No significant challenges have been encountered during the wireframing and initial coding phases. However, I am continuously monitoring potential challenges and am prepared to address any issues that may arise during the development process.	
<b>Now What?</b>  In the upcoming weeks, the focus will remain on code development, user interface refinement.	
<b>Student Signature</b>	

### Supervision & Reflection Journal

Student Name	Nedah Jan Safi
Student Number	X20347946
Course	BSHCYB4
Supervisor	Keith Maycock

**Month: February**

<b>What?</b>  In this month I have worked on creating the QR Code scanner using HTML5 library. Created code in VS Code using HTML, JavaScript, CSS for web page. Have used the HTML5 library to scan the QR Code.
<b>So What?</b>

Understanding how QR Code scanning works and how to do it with HTML5 was tough. I struggled with figuring out how HTML, JavaScript, and the camera on devices all fit together. Making sure everything worked on different web browsers was also tricky.

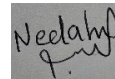
#### Now What?

What can you do to address outstanding challenges?

Using HTML5 and discovered more about what it can do. Learning about how JavaScript interacts with web pages and how to handle events was important for making the scanner work. Also learn about using JavaScript libraries to make web projects better.

Seek feedback from supervisor.

**Student Signature**



### Supervision & Reflection Journal

<b>Student Name</b>	Nedah Jan Safi
<b>Student Number</b>	X20347946
<b>Course</b>	BSHCYB4
<b>Supervisor</b>	Keith Maycock

#### Month: March

##### What?

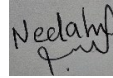
In this month have been working on adding features to QR Code scanner. have used qrcode-parser library to decode the contents of the QR code and retrieve information about its type. Also used Regular expression for URL validation, this ensured that only valid URLs were processed, minimizing the risk of erroneous data interpretation and enhancing the overall robustness of the scanner.

##### So What?

Adding these features to QR Code scanner project it advanced its capabilities and usability. Using the qrcode-parser library to decodes QR codes and retrieve information about its type. Using regular expressions for URL validation to ensure that only valid URLs can process and minimize the risk of erroneous data interpretation and enhancing the overall robustness of the scanner. The challenges still remain is to add the machine learning algorithms to the code. And to make sure it works properly.

**Now What?****What can you do to address outstanding challenges?**

To address the challenges. Have to choose the correct format and libraries of the algorithms, which will help in improvement of the QR Code scanner.

**Student Signature****Supervision & Reflection Journal****Student Name**

Nedah Jan Safi

**Student Number**

X20347946

**Course**

BSHCYB4

**Supervisor**

Keith Maycock

**Month: April****What?**

In this month have been working on adding ML algorithms to detect malicious URL, and have completed the web app. For detecting malicious URL have used lexical feature, logistic regression, and decision tree models. And deployed in JavaScript language for backend and used html and CSS for frontend.

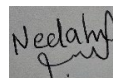
**So What?**

Adding these features and models to QR Code scanner project it advanced its capabilities and usability. Using lexical feature logistic regression, and decision tree to analyses the format of the URL and detect malicious URL.

The next step is to testing, to test if the features are working and obtain all the expedition.

**Now What?****What can you do to address outstanding challenges?**

To address the challenges. Have to choose the correct testing format and using of framework for obtaining good testing result.

**Student Signature**

### 12.1. Other materials used

Any other reference material used in the project for example evaluation surveys etc.