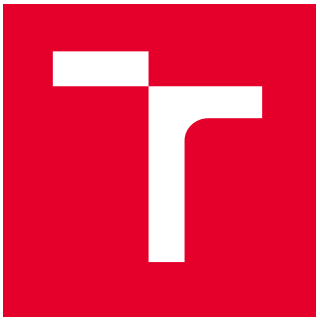


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

SECURE WEB BROWSER FOR SENIORS

BEZPEČNÝ WEBOVÝ PROHLÍŽEČ PRO SENIORY

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR — **Bc. Tarik Alkanan**
AUTOR PRÁCE

SUPERVISOR — **prof. Ing. Dan Komosný, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

Master's Thesis

Master's study program **Information Security**

Department of Telecommunications

Student: Bc. Tarik Alkanan ***ID:*** 221267

Year of study: 2 ***Academic year:*** 2024/25

TITLE OF THESIS:

Secure web browser for seniors

INSTRUCTION:

Create a secure web browser that will be tailored for late elderly and intellectually challenged users. The browser will be easy to operate and will implement protection against fraudulent (phishing) webpages. Detect fraudulent webpages using machine learning. Adapt the created detection model to webpages in the Czech language. In the browser, implement alerts when text is entered into a form on a detected fraudulent webpage. Create the browser in the Python programming language. Publish the results on a GitHub repository under the MIT license.

RECOMMENDED LITERATURE:

According to the instructions of the head of thesis.

Date of project specification: 10.2.2025 ***Deadline for submission:*** 27.5.2025

Supervisor: prof. Ing. Dan Komosný, Ph.D.

prof. Ing. Jan Hajný, Ph.D.
Chair of study program board

ABSTRACT

Tato diplomová práce popisuje vývoj uživatelsky přívětivého webového prohlížeče přizpůsobeného seniorům ve věku 90 let a více. Prohlížeč je součástí operačního systému určeného pro seniory a jeho cílem je zjednodušit používání počítače a zároveň zvýšit bezpečnost na internetu. Prohlížeč vyvinutý v jazyce Python obsahuje funkce, jako je zvětšení textu, základní navigace po webových stránkách a detekce phishingu, která chrání uživatele před podvodnými webovými stránkami. Prohlížeč navíc upozorní určeného opatrovníka, pokud dojde k přístupu na podvodnou webovou stránku, čímž nabízí další úroveň ochrany před podvody. Výsledky tohoto projektu jsou veřejně dostupné v repozitáři GitHub.

KEYWORDS

Phishing, Senior, webový prohlížeč, podvodná stránka, phishingová webová stránka, Strojové učení.

ABSTRAKT

This diploma thesis describes the development of a user-friendly web browser tailored for seniors aged 90 and above. As part of an operating system designed for the elderly, the browser aims to simplify computer use while enhancing safety online. Developed in Python, the browser includes features such as text enlargement, basic web page navigation, and phishing detection to protect users from fraudulent websites. Additionally, the browser notifies a designated guardian if a fraudulent website is accessed, offering an added layer of protection against scams. The outcomes of this project are made publicly available in a GitHub repository.

KLÍČOVÁ SLOVA

Phishing, Senior, Web browser, fraudulent page, phishing website, Machine learning.

Rozšířený abstrakt

Tato diplomová práce představuje návrh a implementaci webového prohlížeče speciálně přizpůsobeného pro seniory ve věku 90 let a více, kteří často čelí značným problémům při adaptaci na moderní digitální technologie. Cílem prohlížeče, který je součástí širší iniciativy Senior Operating System (Senior OS), je zpřístupnit online interakce starším uživatelům, zejména těm, kteří nejsou obeznámeni s běžnými digitálními hrozbami, jako jsou například phishingové útoky, intuitivněji a bezpečněji. Projekt reaguje na rostoucí společenskou potřebu: zatímco internet nabízí řadu výhod - například přístup k informacím, komunikačním nástrojům a digitálním službám -, mnoho seniorů se potýká s jeho složitostí. Tuto propast prohlubuje nedostatečná znalost technologií, zhoršující se zrak, pomalejší motorika a rostoucí sofistikovanost online podvodů. Prohlížeč si je vědom těchto problémů, a proto upřednostňuje jednoduchost ve svém designu, bezpečnost ve svých funkcích a pomoc prostřednictvím zabudovaných podpůrných funkcí. Prohlížeč byl vyvinut v jazyce Python a PyQt5 s využitím modulární struktury, která podporuje dlouhodobou udržovatelnost a škálovatelnost. Jednou z jeho klíčových součástí je sada úrovní ochrany, které určují oprávnění uživatele k prohlížení. Tyto úrovně omezují nebo povolují přístup k vyhledávacímu poli, ručnímu zadávání adres URL a navigaci po odkazech na základě kompetencí uživatele nebo rizikových faktorů. Na nejvyšší úrovni ochrany prohlížeč zcela zakáže ruční zadávání adresy URL a funkci vyhledávání, čímž omezí možnosti vyhledávání. K ochraně před podvodnými webovými stránkami používá prohlížeč hybridní ochranný mechanismus, který zahrnuje jak dynamickou černou listinu URL, tak model strojového učení (ML). Černá listina je synchronizována dvakrát týdně, aby byla zajištěna aktuální ochrana, zatímco ML model je vycvičen k detekci potenciálně škodlivých webových stránek na základě souboru dat známých podvodných a legitimních URL. Implementace tohoto duálního přístupu zvyšuje přesnost detekce a odolnost proti novým technikám útoků. Když senior vstoupí do interakce s podezřelou webovou stránkou a začne zadávat informace, aplikace zareaguje vizuálním upozorněním uživatele prostřednictvím červeně stylizovaného panelu nástrojů a současně upozorní určeného opatrovníka prostřednictvím e-mailu. Tato funkce poskytuje dodatečnou bezpečnostní síť a umožňuje včasný zásah. E-mailová oznámení jsou rozesílána pomocí specializovaného poštovního skriptu poskytovaného důvěryhodnou systémovou komponentou, což zajišťuje spolehlivost a oddělení starostí mezi moduly. Kontaktní informace opatrovníka jsou spravovány prostřednictvím centralizovaného konfiguračního souboru, což zvyšuje udržovatelnost a snadnost nasazení. Pro podporu mezinárodního použití obsahuje prohlížeč vícejazyčnou podporu s lokalizovaným uživatelským rozhraním v češtině, angličtině a němčině. Přístupnost je dále zvýšena díky přizpůsobitelné ve-

likosti písma a zjednodušeným navigačním ikonám. Součástí jsou příručky v každém podporovaném jazyce, které pomáhají seniorům prohlížeč samostatně pochopit a používat. V průběhu vývoje se vyskytlo několik problémů, včetně migrace konfigurační logiky do centralizovaného DataProvideru, integrace správce závislostí Poetry pro zefektivnění sestavení projektu a získání spolehlivé sady phishingových dat pro trénink ML. Tyto překážky byly systematicky řešeny a zdokumentovány v diplomové práci, aby pomohly budoucímu vývoji a nasazení. Veškerý zdrojový kód a dokumentace prohlížeče jsou veřejně dostupné na GitHub, což umožňuje další přispívání ze strany open-source komunity.

ALKANAN, Tarik. *Safe web browser for seniors* . Master's Thesis. Brno: Brno University of Technology, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2024. Advised by prof. Ing. Dan Komosný, Ph.D

Author's Declaration

Author:	Bc. Tarik Alkanan
Author's ID:	221267
Paper type:	Master's Thesis
Academic year:	2024/25
Topic:	Safe web browser for seniors

I declare that I have written this paper independently, under the guidance of the advisor and using exclusively the technical references and other sources of information cited in the paper and listed in the comprehensive bibliography at the end of the paper.

As the author, I furthermore declare that, with respect to the creation of this paper, I have not infringed any copyright or violated anyone's personal and/or ownership rights. In this context, I am fully aware of the consequences of breaking Regulation § 11 of the Copyright Act No. 121/2000 Coll. of the Czech Republic, as amended, and of any breach of rights related to intellectual property or introduced within amendments to relevant Acts such as the Intellectual Property Act or the Criminal Code, Act No. 40/2009 Coll. of the Czech Republic, Section 2, Head VI, Part 4.

Brno ....

author's signature*

*The author signs only in the printed version.

ACKNOWLEDGEMENT

I would like to thank the thesis supervisor, Prof.Ing. Dan Komosny, Ph.D. for his professional guidance, consultation, patience, and suggestions for the thesis. I would also like to express my sincere gratitude to my wife Tetiana for her unwavering support and patience.

Rád bych poděkoval vedoucímu diplomové práce panu prof.Ing. Danovi Komosnému, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Rád bych také vyjádřil upřímnou vděčnost své ženě Tetianě za její neochvějnou podporu a trpělivost.

Contents

List of Figures

List of Tables

Listings

Introduction

The rise of the Internet has significantly transformed communication and information access, marking a new era often referred to as the digital revolution. However, while this shift has benefited much of the population, older users often face difficulties adapting to rapidly evolving technologies. To address these challenges, the Senior Operating System project was launched, with the goal of making daily computer use easier and safer for senior citizens. One of the key components of this project is a senior-friendly web browser, developed in Python, which focuses on both accessibility and security.

The Senior Web Browser (Sweb) introduces several important features tailored to the needs of senior users:

1. **Adjustable text size**, allowing users to read content comfortably.
2. **Multi-language support**, catering to users whose native language is not English.
3. **Enhanced fraud protection**, including dynamic phishing blacklist checks, machine learning-based threat detection, and automatic notifications to guardians when suspicious activity is detected.

A special safeguard mechanism has also been integrated: if a senior user enters sensitive information on a phishing website, an alert is automatically sent to a designated guardian, providing an additional layer of protection and oversight.

This diploma thesis is divided into theoretical and practical parts and is structured into nine chapters:

Chapter 1: The Current Status of the Senior Web Browser - Describes the current functionality of the Senior Web Browser and outlines planned enhancements.

Chapter 2: Provides an overview of phishing and other types of fraud that particularly threaten senior users, along with a discussion of detection techniques .

Chapter 3: Focuses on the development and evaluation of the machine learning model used for phishing detection, including an introduction to artificial neural networks.

Chapter 4: Reviews the main libraries and frameworks utilized in the development of Sweb, and explains the rationale behind their selection.

Chapter 5: Discusses the conceptual design, graphical environment setup, and user interface development aimed at senior accessibility.

Chapter 6: Details the implementation of blacklist-based web security, synchronization of phishing site lists, and the integration of machine learning for URL threat verification.

Chapter 7: Describes the different protection levels provided by Sweb and explains how alerts and user data are securely transmitted to custodians.

Chapter 8: Reflects on the main technical and organizational challenges faced during the project, and how they were addressed.

Chapter 9: Presents multilingual user manuals and explains how the application is distributed and maintained through GitHub.

The thesis concludes with a summary of the project's outcomes, highlighting the solutions implemented to address the identified challenges. The completed work is fully documented and accessible on GitHub.

1 Application Overview and Future Development

This chapter provides an overview of the current state of the web browser for seniors. It describes the basic features of the application and includes a visual representation of the graphical user interface (GUI) to illustrate its functionality and user-friendliness. The second part of the chapter focuses on planned improvements and provides a detailed justification for the proposed changes. It also explains the removal of some previously implemented features and highlights how user feedback, usability considerations and evolving security requirements have influenced the direction of future development. The following sections provide a detailed exploration of the web browser's status and planned development.
Section 1.1 describes the current functionality of the Senior Web Browser, focusing on its main features and graphical user interface.
Section 1.2 outlines the planned enhancements, explains the reasons for these changes, and discusses how user feedback and security needs have shaped future directions.

1.1 Current Functionality of the Senior Web Browser

The web browser for seniors (SWeb) is currently in a functional state. A detailed description of its earlier development can be found in the previous thesis on the VUT website under the name **Webový prohlížeč pro seniory**. The current graphical user interface is shown in Figure 1.1, and the browser includes the following features specifically tailored to the needs of senior users:

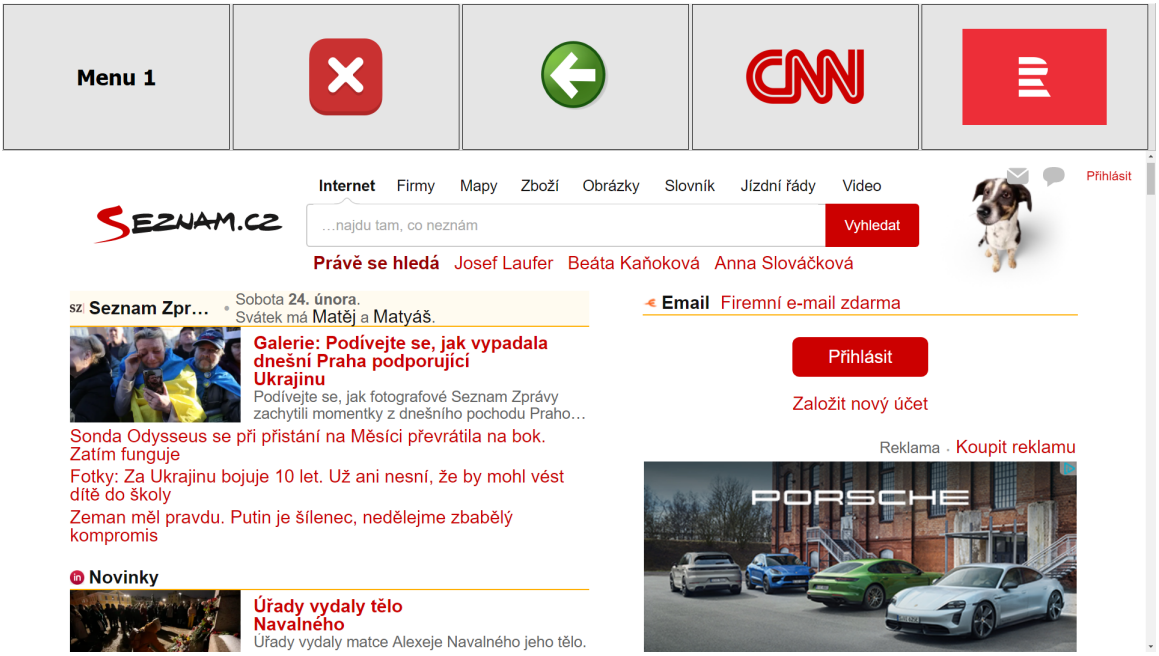


Fig. 1.1: Graphical visual of the previous GUI.

User-Friendly Navigation with Dual Menu Bars

The browser interface is equipped with two intuitive menu bars:

The First Menu Bar contains five buttons: The

- Switch Menu Button: Allows users to toggle between the two menu bars.
- Close button: Close the app
- Back Button: Navigate to the previous page.
- two Website Buttons: Provide quick access to predefined websites.

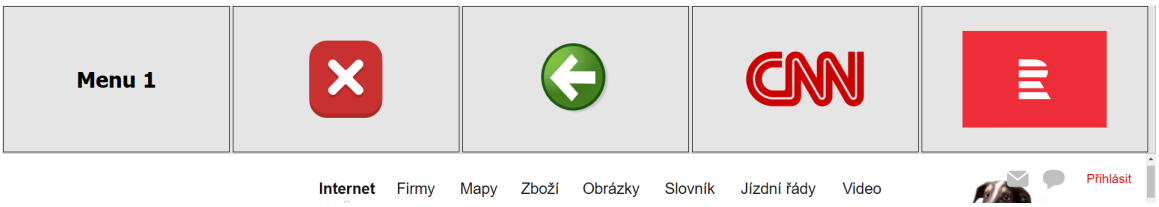


Fig. 1.2: The content of the first menu.

The Second Menu Bar Also has five buttons:

- Switch Menu Button: Toggles back to the first menu bar.
- Three Website Buttons: Connect to additional predefined websites.
- Search Button: Opens a search bar where users can enter text to search for information online.

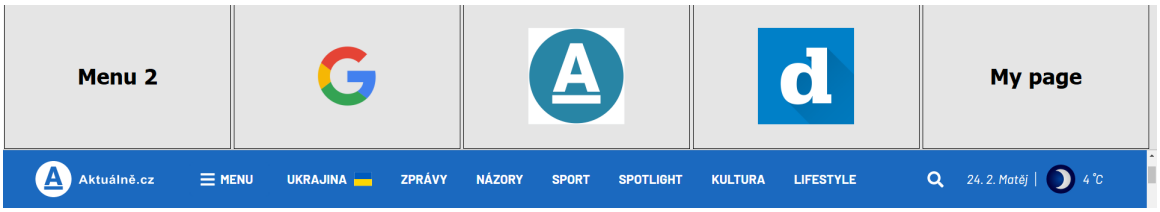


Fig. 1.3: The content of the second menu.

Support for Accessibility and Multilingual Features

- Sound Assistance: The browser integrates sound cues for better navigation and accessibility, assisting seniors in using the app without relying solely on visual input.
- Multilingual Support: Provides a seamless browsing experience by supporting multiple languages, ensuring accessibility for users with different linguistic preferences.

Integrated Mail Client

The application includes a built-in mail client to enhance security. This feature automatically sends alerts to a designated guardian if the user visits a phishing website, ensuring additional oversight and protection.

Configuration and Security Features

The browser uses its own configuration file to store and read default data, such as button assignments and language preferences. Sweb employs a blacklist system to identify and block access to phishing pages.

1.2 Planned Enhancements and Their Justification

The primary objective of this thesis will be to develop a user-friendly and secure web browser tailored for seniors. Several new features and enhancements will be implemented to improve functionality and usability. These updates will address previous limitations and lay the foundation for future advancements. The updated graphical user interface (GUI) of the Senior Web Browser, reflecting these improvements, will be introduced as part of the project, is illustrated in Figure 1.4.

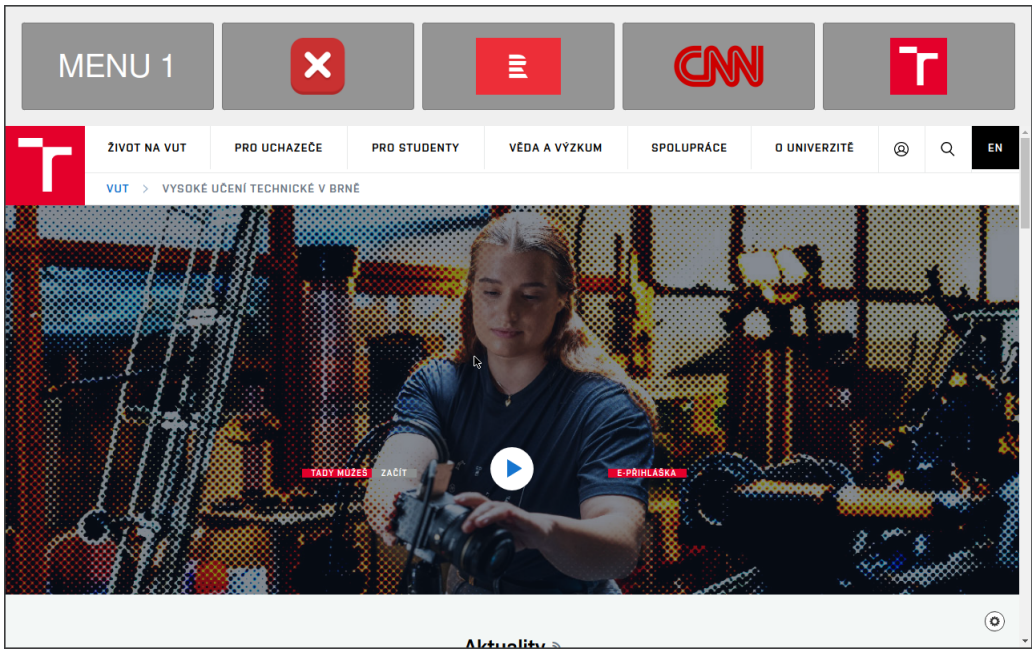


Fig. 1.4: Graphical visual of the new GUI.

Code Modularization

In future development, the source code of Sweb, which is currently structured within a single main class, will be refactored into a modular architecture. Specific functionalities will be separated into distinct modules—for example, a `ui` module to manage the graphical user interface and a `phish` module to handle phishing URL detection and blacklist updates. This planned modularization will significantly enhance code readability, maintainability, and scalability, making the project more adaptable to future enhancements.

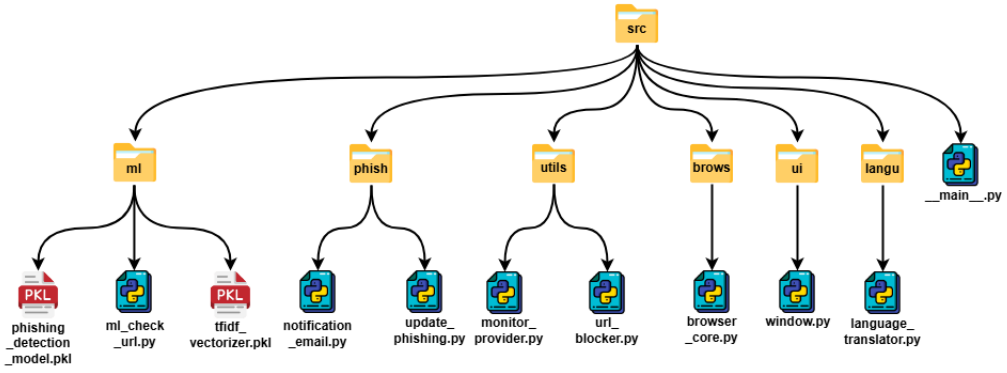


Fig. 1.5: Modules diagram.

Preparation for Neural Network Integration

The groundwork will be laid to enable the integration of a neural network for phishing detection. This advanced feature, aimed at enhancing the browser's ability to identify fraudulent websites, will be trained and incorporated into the application, as shown in figure 6.4. It is planned to serve as a secondary method for URL verification, complementing the existing blacklist-based detection system. The integration and evaluation of the machine learning model will be explored further in the follow-up diploma thesis.

Centralized Configuration Management

The Sweb application will adopt a centralized configuration system through the `_dataProvider` method. This method will consolidate all configuration settings into a single `JSON` file, simplifying management and ensuring consistency across the application. The method is shown in Figure 5.11.

Feature Removals for Improved Stability and Efficiency

In earlier versions of SWeb, two features, sound assistance and a detailed activity log, were included to enhance accessibility and debugging. However, both will ultimately be removed to streamline the application and improve overall performance.

The sound assistance feature, while designed to support users with visual impairments, introduced significant instability and complexity during the integration of SWeb into a live ISO environment. Its removal will lead to a more stable and maintainable application.

Similarly, the logging system, which previously recorded every step of the application's behavior, generated excessively large log files that consumed valuable storage space. Removing this feature will reduce resource usage and result in a more lightweight and efficient application.

2 Fraudulent attacks

This chapter deals with various fraudulent attacks, with a special emphasis on phishing. It highlights phishing detection methods and details the approach taken for this thesis. The following sections provide a comprehensive overview of fraudulent attacks and the methods used to address them.
Section 2.1 introduces the concept of fraudulent attacks and their significance.
Section 2.2 categorizes different types of fraudulent activities that target users.
Section 2.3 focuses specifically on phishing attacks, their mechanisms, and their impact.
Section 2.4 discusses various techniques used for detecting phishing attempts, including the methods implemented in this thesis.

2.1 Introduction to Fraudulent Attacks

Fraudulent online activities are deceptive practices carried out over the internet to take advantage of individuals or organizations. These attacks often involve tricking victims into sharing sensitive information, installing malicious software or making unauthorized financial transactions. Common examples include phishing scams, identity theft, ransomware, and social engineering tactics [1].

Technological advances and increasing reliance on the Internet have made these attacks more sophisticated, targeting unsuspecting users with personalized schemes. Fraudsters use a variety of platforms - emails, websites, social media, and even instant messaging apps - to reach their victims. These activities not only violate privacy but also lead to significant financial loss, emotional distress, and erosion of trust in digital platforms.
Vulnerable populations, particularly seniors, are disproportionately affected by online fraud for several reasons:

- **Limited digital literacy:** Many seniors may lack familiarity with evolving online threats or may not be able to distinguish legitimate communications from scams.
- **Trusting nature:** Seniors often exhibit a higher level of trust in others, which increases the likelihood of falling victim to fraudulent schemes.
- **Targeted scams:** Scammers purposefully create scams to take advantage of seniors' fears, such as fake health offers or financial assistance programs.
- **Lack of immediate support:** Seniors who live independently may not have immediate access to help from tech-savvy family members or friends when they encounter suspicious activity.

The consequences for seniors can be devastating, including financial loss, exposure to additional scams due to compromising information, and emotional trauma.

2.2 Types of Fraudulent Attacks

Social Engineering Attacks

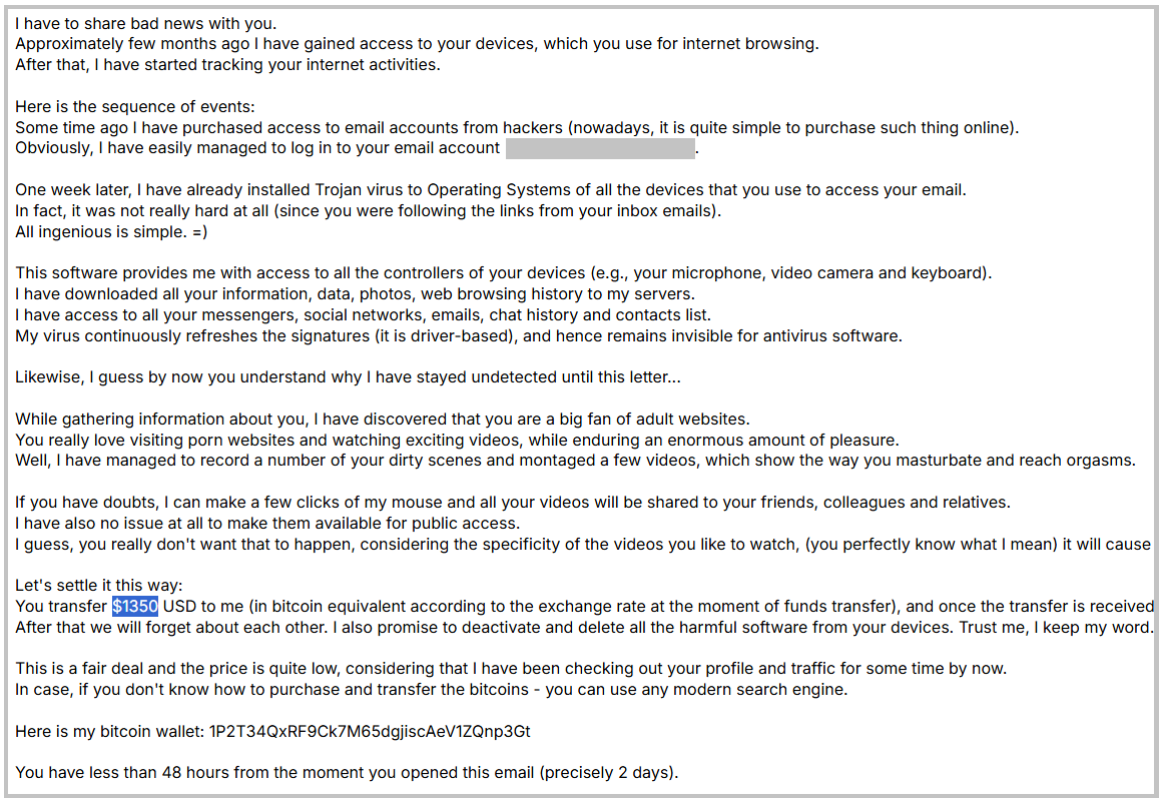
Social engineering attacks rely on psychological manipulation rather than technical hacking to deceive individuals into revealing confidential information or taking certain actions [2].
Manipulation tactics to deceive users:

- Abuse of trust by impersonating a known or authoritative entity (e.g., pretending to be technical support).
- Creating a sense of urgency or panic, e.g. claiming that a bank account has been compromised.
- Acting on emotions such as fear, greed or empathy to lower the victim's guard.

Common techniques:

- Impersonation: Fraudsters impersonate someone they know, such as a friend, relative, or organization.
- Urgent requests: Messages claiming that immediate action is needed, such as transferring money or providing login details.
- Pretext: Creating a fictitious story to gain trust and obtain sensitive information.

Figure 2.1 illustrates a real-world social engineering phishing email, where the attacker employs psychological manipulation techniques to deceive the victim into transferring funds.


I have to share bad news with you.
Approximately few months ago I have gained access to your devices, which you use for internet browsing.
After that, I have started tracking your internet activities.

Here is the sequence of events:
Some time ago I have purchased access to email accounts from hackers (nowadays, it is quite simple to purchase such thing online).
Obviously, I have easily managed to log in to your email account .

One week later, I have already installed Trojan virus to Operating Systems of all the devices that you use to access your email.
In fact, it was not really hard at all (since you were following the links from your inbox emails).
All ingenious is simple. =)

This software provides me with access to all the controllers of your devices (e.g., your microphone, video camera and keyboard).
I have downloaded all your information, data, photos, web browsing history to my servers.
I have access to all your messengers, social networks, emails, chat history and contacts list.
My virus continuously refreshes the signatures (it is driver-based), and hence remains invisible for antivirus software.

Likewise, I guess by now you understand why I have stayed undetected until this letter...

While gathering information about you, I have discovered that you are a big fan of adult websites.
You really love visiting porn websites and watching exciting videos, while enduring an enormous amount of pleasure.
Well, I have managed to record a number of your dirty scenes and montaged a few videos, which show the way you masturbate and reach orgasms.

If you have doubts, I can make a few clicks of my mouse and all your videos will be shared to your friends, colleagues and relatives.
I have also no issue at all to make them available for public access.
I guess, you really don't want that to happen, considering the specificity of the videos you like to watch, (you perfectly know what I mean) it will cause

Let's settle it this way:
You transfer $1350 USD to me (in bitcoin equivalent according to the exchange rate at the moment of funds transfer), and once the transfer is received
After that we will forget about each other. I also promise to deactivate and delete all the harmful software from your devices. Trust me, I keep my word.

This is a fair deal and the price is quite low, considering that I have been checking out your profile and traffic for some time by now.
In case, if you don't know how to purchase and transfer the bitcoins - you can use any modern search engine.

Here is my bitcoin wallet: 1P2T34QxRF9Ck7M65dgjiscAeV1ZQnp3Gt

You have less than 48 hours from the moment you opened this email (precisely 2 days).

Fig. 2.1: Real phishing mail based on social engineering.

Social Engineering Attack Cycle

The Social Engineering Attack Cycle describes the steps an attacker uses to exploit human behavior to gain unauthorized access or manipulate individuals 2.2. Here are the key stages:

1. **Research:** The attacker gathers information about the target (individual, organization, or system). Methods include social media, websites, and public records.
2. **Pretexting (Establishing Trust):** The attacker builds a credible scenario or persona to gain the target's trust. For example, impersonating an IT support technician. Pretending to be a colleague, client, or trusted authority.
3. **Engagement (Exploitation):** The attacker interacts with the target to extract sensitive information or initiate access. Techniques include Phishing emails or calls.
4. **Execution (Attack Deployment):** The attacker uses the obtained information or access to execute the main attack. Objectives include installing malware, stealing data, financial fraud, or gaining deeper network access.
5. **Closure (Covering Tracks):** The attacker completes their mission and exits without raising suspicion. Tactics include deleting traces of their activities and

logging out of accessed accounts or systems.

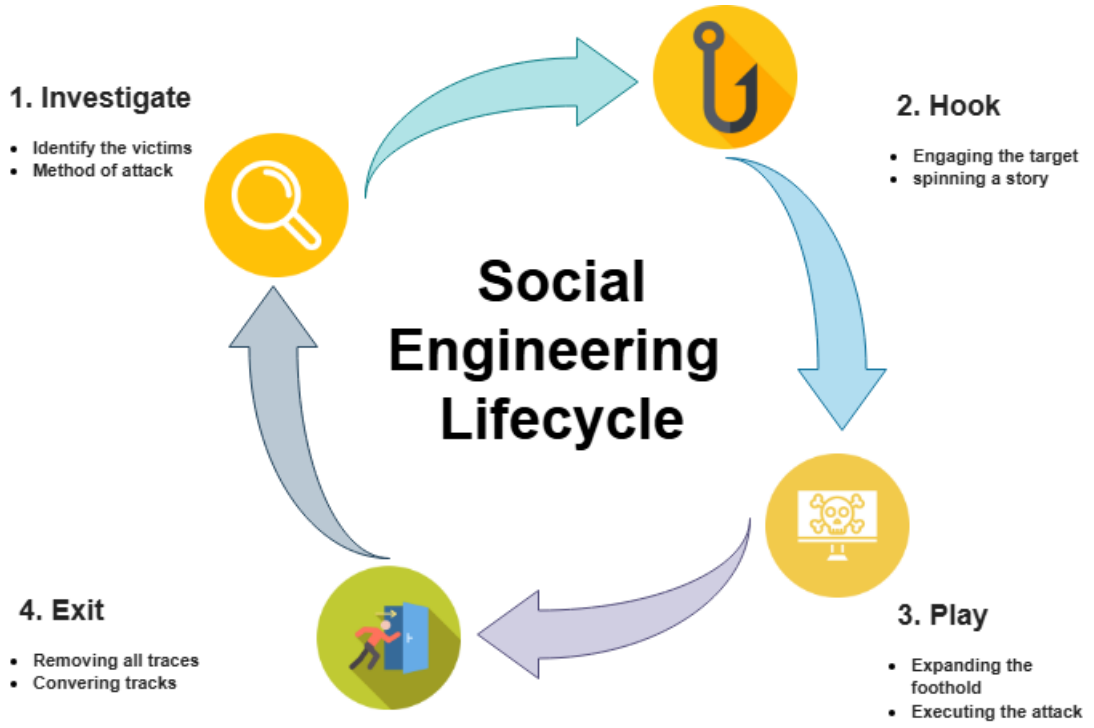


Fig. 2.2: Social engineering attack lifecycle. Reference [2].

Types of Social Engineering Attacks

The Social Engineering Attack can be adapted and executed in various ways, depending on the attack type or context. Here are some types of social engineering attacks, categorized by the method or medium used [2]:

Phishing

Phishing is a type of social engineering attack where attackers send fake emails, messages, or websites that appear legitimate to deceive victims. The primary purpose of these attacks is to steal credentials, and financial details, or infect devices with malware.

Example 2.3 is an email claiming to be from a bank, asking users to verify their accounts by clicking a malicious link, which is a common example of phishing.

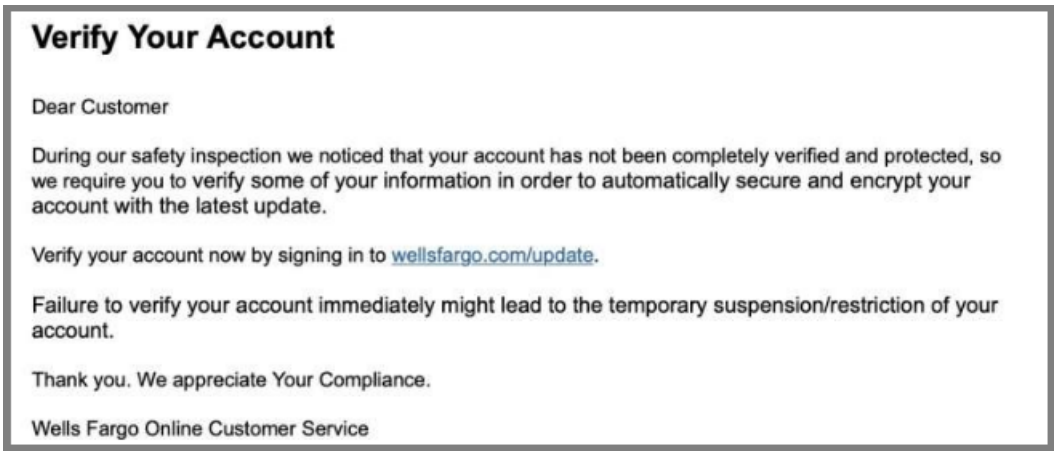


Fig. 2.3: Phishing mail.

Spear Phishing

Spear phishing is a targeted form of phishing tailored specifically to an individual or organization. It involves using personal or organizational information gathered during reconnaissance to create convincing attacks.

Example: An email that appears to come from a CEO, requesting sensitive financial reports or approving a wire transfer, is an example of spear phishing.

Vishing (Voice Phishing)

Vishing is a social engineering attack conducted over the phone, where attackers impersonate trusted entities to deceive victims. This attack often exploits urgency or fear to manipulate individuals.

Example: A scammer pretending to be tech support calls and claims your computer is infected, requesting remote access to "fix" the issue.

Smishing (SMS Phishing)

Smishing is phishing through text messages, designed to trick victims into revealing sensitive information or clicking malicious links. These messages often mimic legitimate organizations.

Example: A text message claiming there is an issue with your bank account and asking you to click a link to resolve it is a common example of smishing.

Pretexting

Pretexting involves creating a fabricated scenario to manipulate the victim into revealing sensitive information or performing actions. The attacker establishes trust

by impersonating a legitimate entity or person.
Example: An attacker pretending to be IT support contacts an employee and asks for their login credentials to fix a "system issue."

Baiting

Baiting uses enticing offers or items to trick victims into taking harmful actions. The "bait" can be physical (e.g., USB drives) or digital (e.g., free downloads).
Example: An attacker leaves an infected USB drive labeled "Confidential Data" in a public area, hoping someone will plug it into their computer.

Quid Pro Quo

Quid pro quo attacks involve offering something valuable in exchange for sensitive information or access. Attackers exploit curiosity or need to manipulate victims.
Example: A scammer offers a "free antivirus upgrade" in exchange for login credentials.

Watering Hole Attacks

Watering hole attacks compromise websites frequently visited by the target group, infecting the site with malware. Victims unknowingly download the malware when accessing the site.
Example: Injecting malicious code into an industry-specific forum used by professionals to infect their devices with spyware.

Reverse Social Engineering

In reverse social engineering, attackers create a problem for the victim and then pose as a helper offering a solution. This tactic manipulates victims into voluntarily providing access or information.
Example: An attacker spreads malware and later contacts the victim as "tech support," offering to fix the issue for a fee.

2.3 Phishing Attacks

In this section, a detailed examination of phishing attacks will be conducted, as the focus of our app is on detecting phishing pages.

How Phishing Works

Phishing attacks exploit human trust and lack of cybersecurity awareness to trick users into revealing sensitive information [3]. The process usually proceeds in the following steps:

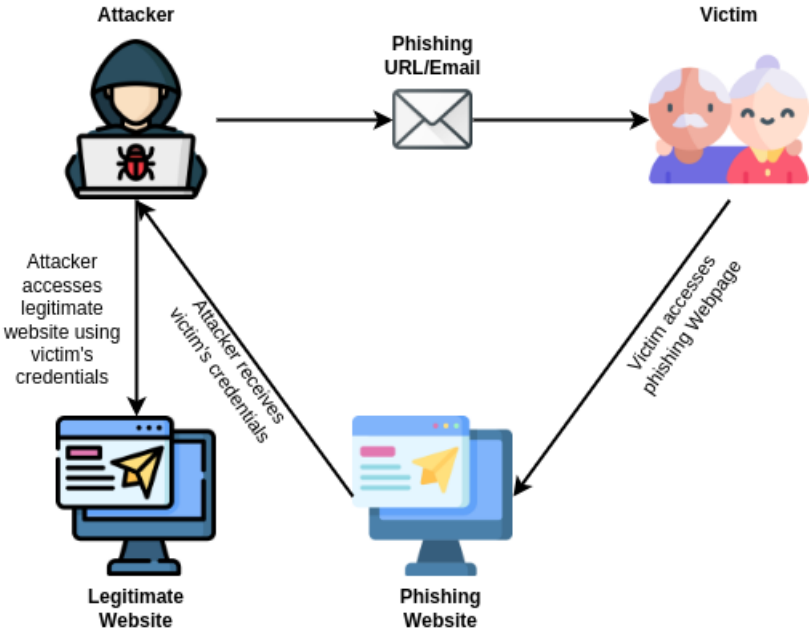


Fig. 2.4: Phishing attack diagram [3].

Breaking down a phishing attempt step by step:

1. **Planning the attack:**
   The attacker identifies the target group and designs a phishing campaign tailored to their characteristics (e.g., seniors).
   Creates fake websites, emails, or messages that mimic legitimate entities such as banks, social platforms, or government agencies.
2. **Launch an attack:**
   Attack: a phishing email or message is sent to the target, often containing a malicious link, attachment, or form.
   The message uses tactics such as urgency ("Your account will be blocked") or fear ("Suspicious activity has been detected on your account") to prompt immediate action.
3. **Victim Interaction:**
   The victim clicks on a link or downloads an attachment that either: Redirects to a fake website where sensitive information is collected.
   Installs malware that records their login credentials or other sensitive information.

4. **Misuses the stolen data:**
   An attacker uses the collected data to access accounts, make fraudulent transactions, or sell information on dark web marketplaces.

Common signs of phishing:

- Suspicious URLs: Domain names that look similar to legitimate domains (e.g. "amaz0n.com" instead of "amazon.com").
- General greetings: Use of non-personalized greetings, such as "Dear Customer".
- Grammatical and spelling errors: Poorly written messages that do not match official communications.
- Urgent language: Phrases such as "Act now", "Check immediately" or "Your account will be suspended".
- Unusual Sender Addresses: Email addresses that do not match the official domain (e.g. "support@securebankxyz.com").

Consequences of Phishing

Phishing attacks can have serious short- and long-term effects on victims, especially seniors.

Risks to personal data:

- Sensitive information such as social security numbers or medical records can be misused for identity theft.
- Once data is stolen, victims become more vulnerable to future attacks (e.g. fraudsters selling data to other criminals).

Financial loss:

- Unauthorized bank transactions, credit card fraud, or even depletion of retirement accounts.
- Victims may have difficulty recovering stolen funds, especially if they act too late.

Emotional stress:

- Seniors often feel guilt or embarrassment after being a victim of fraud.
- Fear of losing their independence or being seen as incompetent may prevent them from reporting incidents.

2.4 Phishing Detection Techniques

Phishing detection techniques can be broadly divided into traditional and advanced techniques using machine learning. Traditional methods, such as blacklists and heuristic-based detection, rely on predefined patterns or lists of known threats.

While these methods are effective for known phishing attempts, they often struggle with new and evolving attacks.

Traditional methods

Traditional phishing detection methods have been the basis of the fight against phishing attacks for many years. These include:

- **Blacklists:** Maintained lists of known phishing websites that are regularly updated to block access to malicious sites. Blacklists are effective for previously identified threats but can fail to detect new phishing or zero-day attacks.
- **Heuristic approaches:** These methods rely on manually created rules or patterns, such as checking for suspicious URL structures, mismatched domains, or spoofed email addresses. Heuristic approaches allow for fast detection but are often not scalable and adaptable enough to evolving phishing tactics.
- **Content-based filtering:** Analyzing the content of emails or web pages for keywords, links, or HTML patterns that are commonly used in phishing. However, this method can lead to a high false positive rate because attackers obfuscate the content to avoid detection.

In our work, we will use blacklists as one component of the detection framework and benefit from their effectiveness against known threats.

Advanced Techniques Using Machine Learning

To overcome the limitations of traditional methods, we will integrate machine learning (ML) techniques into our approach. This method will be explained in greater detail in the DP thesis. These methods include:

- Supervised Learning: Using labeled datasets of phishing and legitimate websites to train models that can classify unknown sites.
- Feature Extraction: Identifying characteristics such as domain age, URL length, and SSL certificate validity to distinguish between phishing and legitimate websites.
- Anomaly Detection: Detecting unusual patterns in user behavior or network traffic that may indicate phishing attempts.

By combining traditional blacklist methods with advanced machine learning techniques, our thesis will propose a hybrid detection system aimed at improving accuracy, scalability, and adaptability.

3 Machine learning

Machine learning serves as a powerful tool for detecting fraudulent activities and is integrated into the application as an additional layer of protection for the senior user's guardian [28]. The following sections will explore the historical development and practical applications of machine learning in this context. After outlining the evolution of the field, the chapter will focus on key machine-learning concepts relevant to this work, including artificial neural networks and methods for evaluating model accuracy. The following sections delve deeper into the role of machine learning in the project.
Section 3.1 outlines the development and evolution of machine learning and its relevance to cybersecurity.
Section 3.2 provides an overview of artificial neural networks, with a focus on concepts applied in this thesis.

3.1 Development of the Machine Learning

This section explores the history and evolution of machine learning, tracing its journey from an early conceptual vision to its initial implementations. It concludes with an overview of the current advancements in neural network development and offers a forward-looking perspective on potential future directions.

Alan Turring

A pivotal moment in the early development of machine learning and artificial intelligence can be traced back to 1950 when the renowned British scientist Alan Turing published his seminal paper "Computing Machinery and Intelligence." In this influential work, Turing introduced what would later become known as the Turing Test, a foundational concept in the philosophy of artificial intelligence [27].

The central question posed by Turing was, "Can machines think?" To explore this, he proposed an experimental framework, originally presented as a variation of the so-called Imitation Game, Figure 3.1. In the original version of the game, an interrogator (observer) communicates with two hidden participants, typically a man and a woman via text. One of the participants attempts to imitate the other's gender, while the observer tries to determine who is who. Turing adapted this concept by replacing one of the human participants with a machine. The machine's objective was to respond in such a way that the observer could not reliably distinguish it from a human. If the machine succeeded in convincing the observer of its humanity, it was said to have demonstrated a form of intelligence.

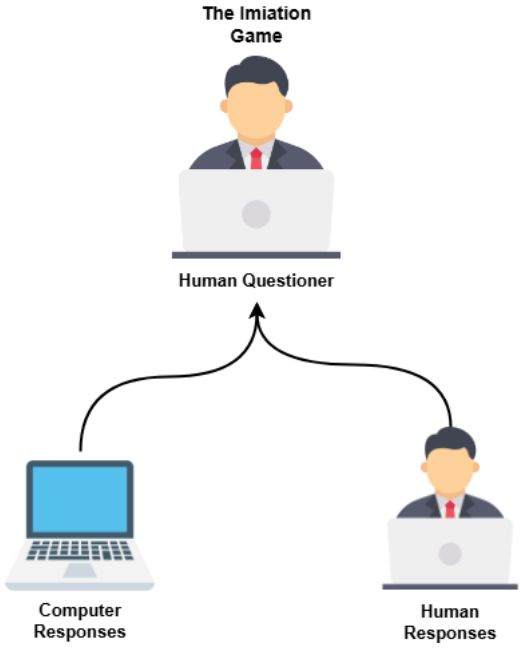


Fig. 3.1: The Imitation game [27].

This conceptual experiment marked a significant shift in the way researchers approached the idea of intelligent machines. Turing's proposal not only initiated academic discourse on the cognitive capabilities of machines but also laid the groundwork for later exploration into machine learning, where systems aim to replicate or simulate aspects of human intelligence such as reasoning, learning, and adaptation.

The first neural network

The concept of neural networks predates modern implementations and can be traced back to 1943 when Warren McCulloch and Walter Pitts introduced a foundational model in their study on the logical calculus of nervous activity [26]. Their work laid the theoretical groundwork for artificial neurons, presenting a simplified, algorithmic representation of how biological neurons might operate, Figure 3.2.

A significant advancement came in 1957 with the development of the perceptron by Frank Rosenblatt. The perceptron is widely regarded as the first neural network model with functionalities resembling those of contemporary neural networks. Building upon the McCulloch-Pitts model, Rosenblatt's perceptron introduced a mathematically defined learning rule, enabling the system to adjust its parameters based on input data, an essential feature for modern machine learning systems.

The perceptron drew considerable attention and marked the beginning of practical neural network research. In 1958, Rosenblatt and his collaborators developed

the first hardware implementation of a neural network, effectively an early neural computer. Its initial application was in optical character recognition, using a photosensitive input device to identify patterns and letters.

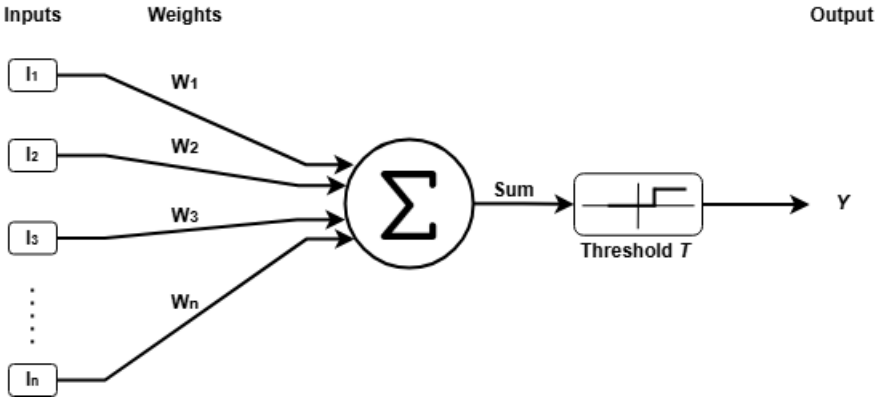


Fig. 3.2: The foundational neural model by McCulloch and Pitts [26].

Modern Development of Neural Networks

In recent years, the popularity of machine learning has surged, driving significant advancements in algorithms and the development of artificial intelligence tools that are increasingly accessible to non-expert users. This growing interest has been particularly evident in the rapid evolution and widespread adoption of deep neural networks, which have enabled breakthroughs in complex tasks such as image recognition, natural language processing, and human-like behavior modeling.

One of the most impactful areas of application has been healthcare, where machine learning models are used to analyze large datasets for advanced diagnostics. These models can identify patterns in patient data, enabling the detection of rare or previously unknown conditions by comparing them with known cases. Such data-driven insights support more accurate and timely medical decisions [29].

Another important field benefiting from machine learning is robotics. In industrial settings, robotic arms are often guided by trained models that enable them to perform precise manipulations. In consumer robotics, autonomous systems such as robotic vacuum cleaners utilize sensor data to map their environments. Based on this data, machine learning algorithms optimize their navigation paths to clean efficiently while minimizing redundant movement.

These modern applications demonstrate not only the growing capabilities of neural networks but also their adaptability across a range of domains, from everyday devices to critical health diagnostics.

3.2 Overview of Artificial Neural Networks

Artificial neural networks (ANNs) play a foundational role in modern machine learning algorithms. They serve as the core mechanism that enables models to learn patterns, make predictions, and improve over time based on input data. Inspired by the structure and functioning of the human brain, ANNs are designed to mimic the way biological neurons process and transmit information.

While the analogy is conceptual, ANNs can be viewed as simplified electronic models of neural activity in the brain. Through interconnected layers of artificial neurons, these systems are capable of identifying complex patterns within data, generalizing from examples, and adapting to new inputs. Their ability to learn from labeled or unlabeled data makes them especially powerful in a wide range of applications—from image and speech recognition to medical diagnostics and autonomous systems.

Key Components of Neural Networks

At the heart of every artificial neural network lies a set of interconnected units known as neurons or perceptrons. These artificial neurons are inspired by biological neurons in the human brain and are designed to replicate, in a simplified form, the process of receiving, processing, and transmitting information. Each neuron acts as a computational node that processes input data and passes the result to other neurons in the network.

Weights and Biases

Two fundamental parameters that govern the behavior of artificial neurons are weights and biases. These parameters play a crucial role in determining how input signals are transformed as they pass through the network.

- **Weights** represent the strength of the connection between neurons. A higher weight indicates a stronger influence from one neuron to another, allowing the network to emphasize more important features in the data.
- **Biases** provide each neuron with a trainable constant value, allowing the network to shift the activation function and improve its flexibility in fitting the data.

A simplified way to visualize this is to consider each neuron N having its own associated weight W and bias B. Connections between neurons are represented by weighted links, the stronger the weight, the greater the signal that is passed. The bias adds an offset to the neuron's output, enabling the model to better fit complex datasets.

In essence, weights control the flow of information, while biases adjust the activation threshold. Together, they are continuously optimized during the training process to reduce error and improve the network's predictive performance.

Activation Functions

Another critical component of neural networks is the activation function, which introduces non-linearity into the model. Without activation functions, a neural network would behave like a linear regression model, limiting its ability to capture complex patterns.

The activation function determines whether a neuron should be "activated" based on the weighted sum of its inputs and the bias. This decision impacts whether and how information is propagated to subsequent layers. Common activation functions include ReLU (Rectified Linear Unit), sigmoid, and tanh, each offering different properties suited to various tasks.

By incorporating non-linearity through activation functions, neural networks are capable of learning intricate relationships within the data, enabling the construction of deep models that can handle tasks such as image classification, language modeling, and pattern recognition.

4 Used Tools

This chapter describes in detail the tools used to build the web browser and train the neural network for phishing detection, including the graphical library, system library, and other key components. Figure 4.1 shows an overview of the libraries used. The following sections provide a detailed overview of the technologies utilized in the project.

Section 4.1 introduces the graphical libraries used for building the user interface.

Section 4.2 presents the `WebEngine` framework, which powers web content rendering.

Section 4.3 describes the library responsible for handling HTTP requests and responses.

Section 4.4 focuses on the file interaction library used for data storage and retrieval.

Section 4.5 outlines the system libraries essential for various backend functionalities.

Section 4.6 discusses the machine learning libraries used for model development and integration.

Section 4.7 explains the adoption of `Poetry` as the project management tool.

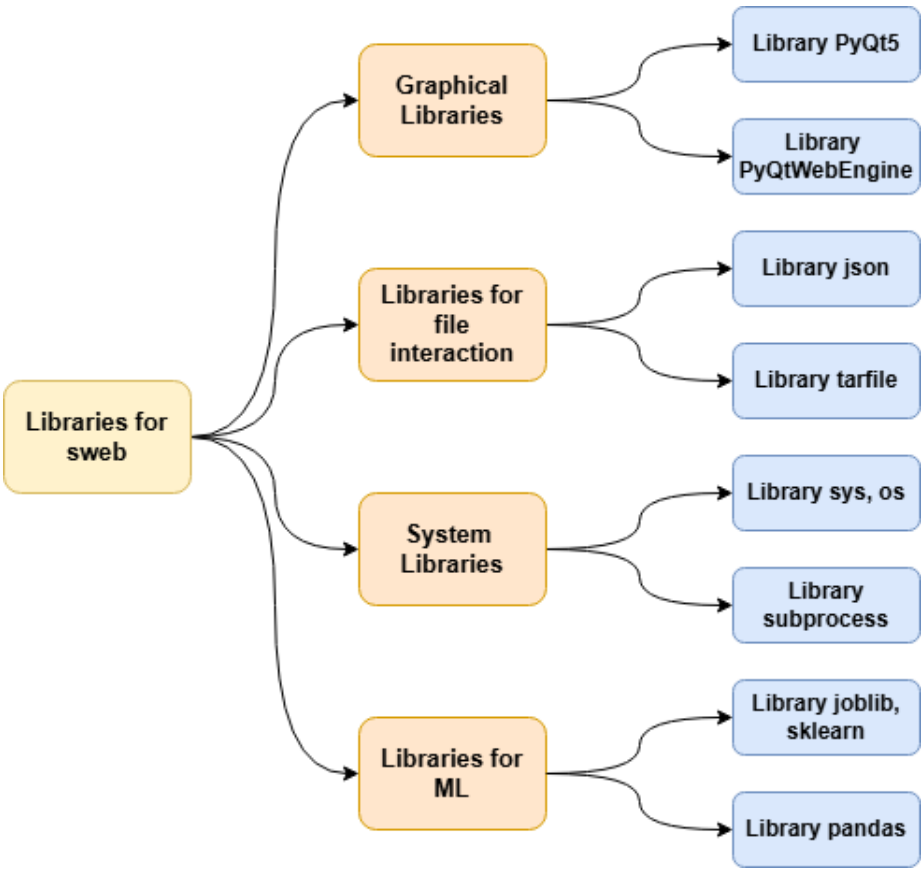


Fig. 4.1: Overview of the libraries used to create the sweb.

4.1 Graphic Libraries

The graphical library plays a key role in creating a user-friendly, senior-friendly web browser interface. It facilitates the display of web content in the browser, allows seamless interaction between different elements of the application and enhances overall usability.

In this work, `PyQt5` and `PyQtWebEngine` libraries are used to implement these features. `PyQt5` provides tools for creating robust and visually appealing graphical user interfaces, while `PyQtWebEngine` integrates web content rendering capabilities [5]. Together, they enable the development of a web browser that is accessible, functional, and optimized for the needs of seniors.

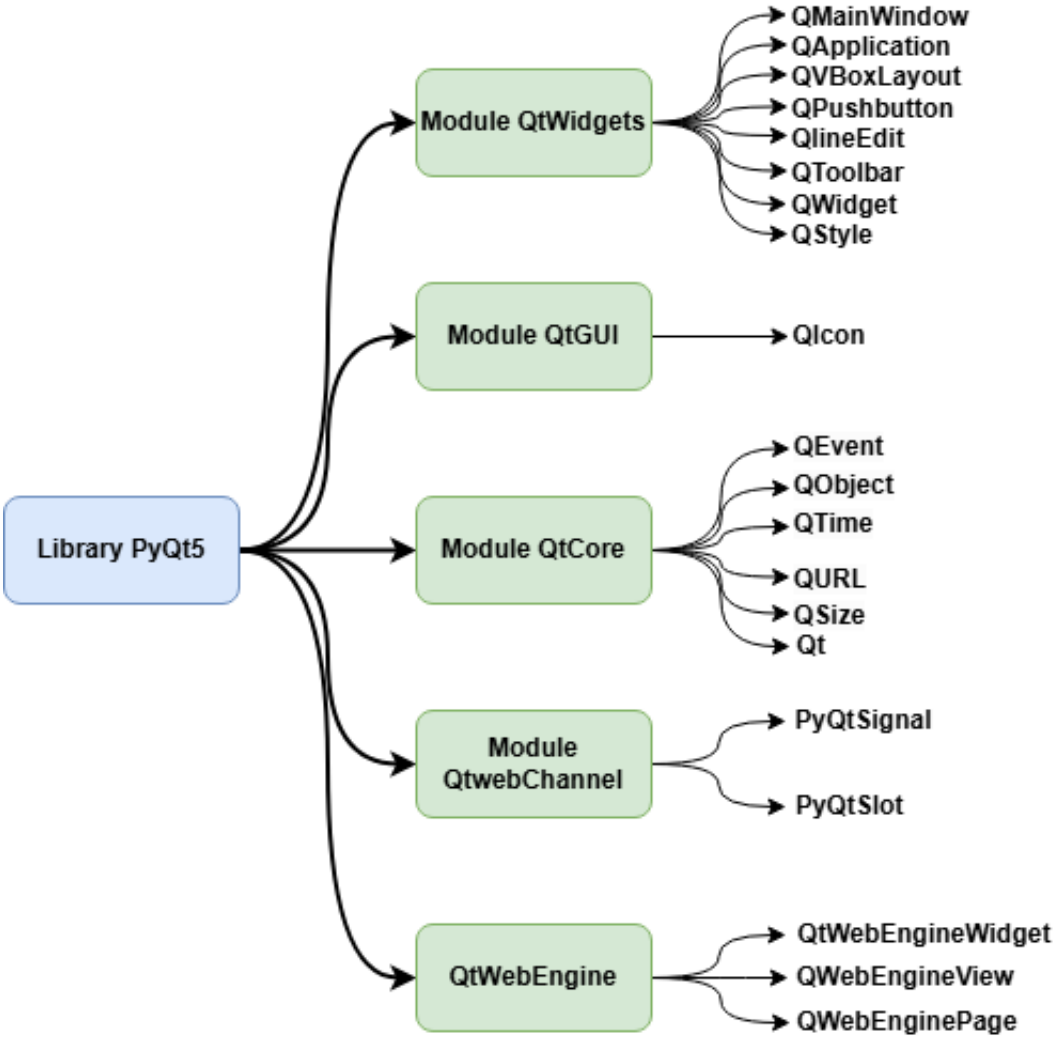


Fig. 4.2: The used PyQt5 modules in SWEB.

Graphical User Interface Development with PyQt5

The `PyQt5` library is an essential tool for developing graphical user interfaces (GUIs) and offers a wide range of features that facilitate the creation of intuitive and visually appealing applications [4]. Its flexibility and ease of use make it ideal for designing complex interfaces, such as those needed for a senior-friendly web browser.

In addition, the `PyQtWebEngine` library is used to seamlessly integrate web content into the browser, complementing the interface design capabilities of `PyQt5`. Together, these libraries provide a comprehensive solution for creating a fully functional and user-friendly web browsing application.

Benefits of Using `PyQt5`:

- Cross-Platform Compatibility: `PyQt5` supports development on Windows, macOS, and Linux, ensuring broad usability for the application.
- Rich Widget Toolkit: It offers a diverse range of pre-built widgets that simplify interface development and customization.
- Integration with Python: `PyQt5` combines the simplicity of Python with the power of the `Qt` framework, making it highly efficient for GUI development.
- Customizability: The library supports extensive customization, allowing developers to create interfaces that meet specific user needs.
- Community and Documentation: A large community of developers and detailed documentation provide valuable resources for troubleshooting and optimization.

Both `PyQt5` and `PyQtWebEngine` will be discussed in greater detail in the following chapter, highlighting their contributions to the development of this thesis project.

The QtWidgets module

The `QtWidgets` module plays a key role in the development of graphical user interfaces (GUIs). It provides a comprehensive collection of user interface (UI) elements that are both highly customizable and easy to use. This versatility makes it particularly suitable for creating interfaces tailored to specific user groups, such as older users. Features such as easy-to-read text and larger button displays, along with simplified navigation, fit well with the accessibility needs of seniors [6].

Several classes from the `QtWidgets` module were used in this work, including `QApplication`, `QMainWindow`, `QToolbar`, `QPushButton`, and others. These classes are imported into the project using the Python import command. Listing 4.1 displays the command that is used in the web browser for seniors.

Listing 4.1: Importing the necessary classes from the `QtWidgets` module.

```
1 from PyQt5.QtWidgets import QMainWindow, QApplication, QStyle, QLabel,
    QVBoxLayout, QHBoxLayout
2 from PyQt5.QtWidgets import QLineEdit, QPushButton, QToolBar, QWidget,
    QSizePolicy
```

QApplication class

The `QApplication` class serves as the basis for every `PyQt5` application. It manages the basic settings and controls the flow of the graphical user interface (GUI). This class provides the basic infrastructure for event handling and communication between the various components of the application [7].

In practice, this means that the `QApplication` is responsible for initializing and finalizing the application, handling user interactions, and performing cleanup tasks after the application has finished. For a web browser designed specifically for older users, the `QApplication` class plays a key role in ensuring that the application remains responsive to user interactions.

QMainWindow class

The `QMainWindow` class is responsible for creating the primary application window. It serves as a central framework that integrates key standard window components such as the toolbar, status bar, and a central area capable of hosting various application elements. This design provides a structured and modular way of organizing the user interface.

For a web browser adapted to seniors, the `QMainWindow` class is particularly valuable. It allows for an intuitive arrangement of the user interface elements and ensures that the interface is easy to navigate even for users with limited technical experience. The clear separation of functional areas contributes to an accessible and user-friendly environment for the seniors.

QToolBar class

The `QToolbar` class is a versatile component that can be added to the main window to provide quick access to frequently used application elements or web functions. It is highly customizable, making it ideal for addressing the specific needs of older users. For example, toolbar elements can be designed with increased visibility, such as larger icons and clear labels. In addition, the `QToolbar` class allows you to reposition the bar within the main application window, improving accessibility for users with limited mobility or dexterity.

In this application, the toolbar has been implemented to accommodate the main web browser buttons. By clicking on these buttons, older users can go directly to specific pre-configured web pages, providing a straightforward and intuitive way to access basic web resources.

QWidget class

The `QWidget` class serves as a basic building block for creating graphical user interfaces in `PyQt5`. It provides basic functionality for managing various UI components such as buttons, windows, and other interactive elements. This class encapsulates the behavior and properties of these components and offers flexibility in customizing their appearance and functionality.

In this application, the `QWidget` class is used to design and arrange interface elements in application windows. Its versatility allows for an intuitive arrangement of user interface components and ensures that the interface remains accessible and easy to use for older users. By leveraging the `QWidge` class, the application windows were structured to support clear navigation and usability and address the specific needs of the target audience.

QPushButton class

The `QPushButton` class is a basic component for creating graphical user interfaces. It allows the creation of buttons that can perform actions when pressed or clicked. This class is essential for designing clear and intuitive interfaces that allow users to directly initiate specific actions. Buttons can display both text and icons, which increases their functionality and makes them more visually informative [8].

In this application, the `QPushButton` class was used to create buttons that provide seniors with direct access to key functions.

QStyle and QLabel classes

The `QStyle` class is used to manage the appearance of graphical user interfaces, including customizing standard icons. This class provides the flexibility to customize icons and other visual elements to be visually appealing and easily recognizable.
The `QLabel` class is a universal component that is used to display text or images in a graphical user interface. It is commonly used to provide explanations or additional information about other interface elements such as buttons [9].

In this application, the `QLabel` class was used to display text descriptions that explain the function of specific buttons.

QVBoxLayout class

The `QVBoxLayout` class is a layout manager that vertically arranges widgets in the user interface. It is particularly useful for clear and structured alignment of elements, which increases ease of use and readability [10].

In this application, the `QVBoxLayout` class was used to vertically arrange icons and text, creating a logical and accessible layout. The layout was adapted to the needs of older users by placing frequently used buttons in visible and convenient locations.

QLineEdit class

The `QLineEdit` class is a widget that provides a single-line text field for user input. It allows users to enter text, so it is suitable for a variety of use cases, such as password input fields or input for search queries [11].

In the context of this application, the `QLineEdit` class is used to create a text input field on a web browser interface. Users can either enter the URL of the web page they want to visit or submit a search query.

QtGUI module

The `QtGui` module is a core component for developing graphical elements within a graphical user interface. It provides various tools for creating and managing visual elements, including the `QIcon` class, which allows the creation of custom icons to enhance the user experience [12].

In this application, the `QtGui` module plays a key role in designing clear, scalable icons that remain readable at any size - an essential feature for senior users. The scalability of the icons ensures that they are easily recognizable even when resized

QtCore module

The `QtCore` module is a non-GUI component that plays an important role in controlling the basic operations of the application. It provides basic functionality for tasks such as file operations, event processing, date and time manipulation, and implementation of the signal and slot mechanism [12].

In the context of this web browser, the `QtCore` module ensures that the application remains interactive, which is especially important when designing for senior users. This interactivity is facilitated by efficient event processing that allows the browser to respond promptly to user actions such as clicks or form submissions. The necessary classes from the `QtCore` module that are used in this web browser are imported as listed.

In addition, the signal and slot mechanism allows seamless communication between different elements of the application. This is very important in a web browser, where common actions such as clicking links, submitting forms, or updating pages are very important. The `QtCore` module provides several key classes used in this work, including `QEvent`, `QUrl`, `Qt`, `PyQtSignal`, and others. Listing 4.2 displays the command that is used in the web browser for seniors.

Listing 4.2: Importing the necessary classes from the `QtCore` and `QtGui` modules.

```
from PyQt5.QtCore import QEvent, QUrl, Qt, QTimer, QSize
from PyQt5.QtCore import pyqtSignal, QObject, pyqtSlot
from PyQt5.QtGui import QIcon, QKeyEvent
```

QEvent class

The `QEvent` class is used to manage events in the application and captures various user actions when interacting with the application. These events trigger corresponding responses or actions within the application, ensuring that the interface responds dynamically to user input [13].

In the context of this web browser, the `QEvent` class plays a vital role in providing responsive and interactive feedback to older users. By capturing and responding to user actions, such as clicks, keystrokes, or other interactions.

QURL class

The `QUrl` class is used to correctly interpret and process web addresses (URLs). Using this class, users can efficiently and safely navigate web pages, which is essential for protecting against fraudulent or malicious websites. In the context of this work, the `QUrl` class is specifically used to remove the default URL parameter from the main window, ensuring that the browser navigation remains customizable and safe.

Qt class

The `Qt` class provides a global identifier and useful functions that define the properties of objects in the application. These functions are used throughout the application to maintain consistency of behavior and appearance [13].

QTime class

The `QTime` class provides functions for tracking and managing time in an application. It allows developers to manipulate and retrieve time values, ensuring accurate

execution of time-sensitive operations. In this application, the `QTime` class is used to manage time-related functions [13].

QSize class

The `QSize` class is responsible for managing the dimensions of windows and application elements. It provides functions for loading and adjusting the dimensions of UI components, ensuring that the layout remains consistent and visually organized. In this application, the `QSize` class is used to adjust the size of various windows and UI elements [13].

QObject class

The `QObject` class is the base class for all `Qt` objects and allows them to respond to various events in the application. This includes user actions, such as input events that are captured and processed by the application [13].

When creating object hierarchies, `QObject` simplifies the management of application components by providing an ordered structure. This hierarchical system is essential for organizing and managing elements within a web browser and provides a clear and efficient framework for application components. This hierarchical system is necessary for the arrangement of elements in the web browser and is shown in Figure 4.3.

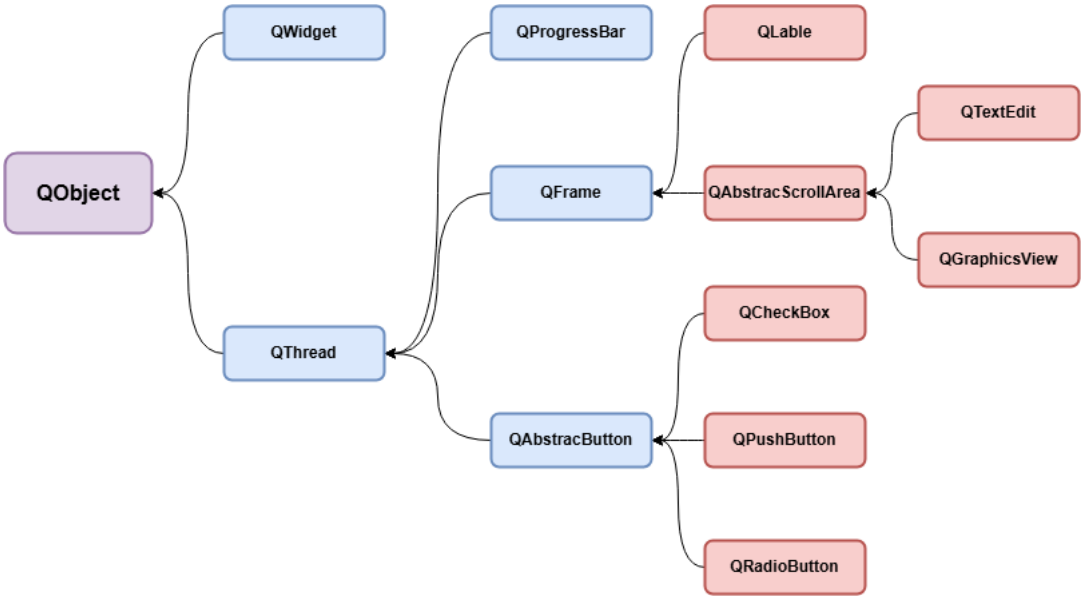


Fig. 4.3: Hierarchical system for arranging `Qobject` elements.

QtWebChannel module

The `QtWebChannel` module provides a key capability to enable interaction between `Qt` applications and `HTML/JavaScript` languages running in the web engine. The basic mechanism of `QtWebChannel` involves exposing C++ classes derived from the `QObject` class to `HTML/JavaScript`. This creates a two-way communication channel where applications and web content can initiate interactions [14].

In this work, the `QtWebChannel` module is used to capture text entered by the user into a form field on a potentially fraudulent web page. This functionality is achieved by modifying the properties of the text field and capturing a signal when the user completes the text entry.

In addition, the `QtWebChannel` module allows dynamic changes to the content of the web page displayed in the browser. The ability to adjust the size of the text is particularly beneficial for the seniors, as it simplifies navigation and improves readability.

PyQtSignal signal

`PyQtSignal` is a powerful feature in `PyQt5` that allows communication between different objects in the browser and also the customization of custom signals. This feature provides real-time updates and interactions within the web browser [15].

For example, when users click on a button to navigate to the home page or open a new tab, the `PyQtSignal` class ensures that these actions immediately trigger appropriate responses in the application. In addition, it can be used to notify the browser to perform specific tasks, such as updating the view when a page has finished loading or when new content is made available.

signal pyqtSlot

`PyqtSlot` is part of the signal and slot mechanism in `PyQt5`. This mechanism works by marking functions as slots, which can then be attached to signals. Using `pyqtSlot` improves the performance and security of the application [15].

In the context of a web browser, this precision is essential to ensure that the correct actions are triggered in response to user interactions. For example, attaching a signal to a specific slot allows the browser to execute tasks efficiently, without unnecessary delays, which improves the overall responsiveness and stability of the application.

4.2 WebEngine Framework

The web kernel `PyQtWebEngine` plays a key role in providing the web browsing functionality in `PyQt5`. It acts as the core component that renders and displays web content and allows the application to load and interact with web pages seamlessly. The web core is responsible for rendering the `HTML`, `CSS`, and `JavaScript` necessary to display dynamic, interactive and visually rich content in the browser [17].

In the context of this web browser application, the `QtWebEngine` module serves as the bridge between the user interface and the web content, embedding web pages within the application window. This integration allows users to browse the web, interact with web forms, and view multimedia content directly within the `PyQt5` application, without the need for external web browsers.

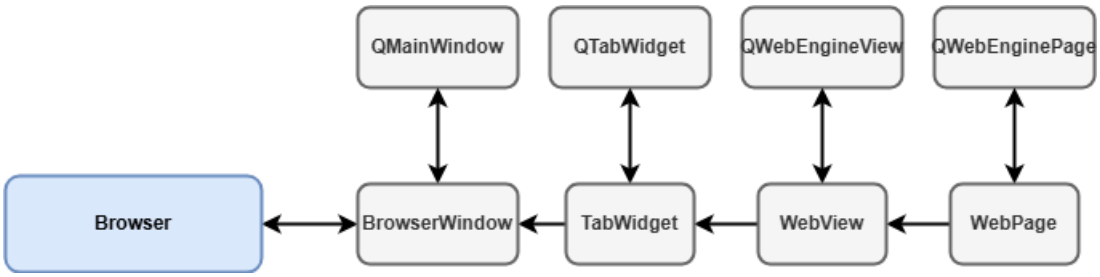


Fig. 4.4: The `QWebEngineView` class and the `QWebEnginePage` class. Reference [25].

The `QtWebEngineWidget` specifically is used to display web pages within a `QWidget-based` layout, providing a seamless and integrated browsing experience.

The QtWebEngineWidget module

The `QtWebEngineWidget` module provides a number of classes for inserting web content into a web application. It is an essential component for the development of modern web browsers [17].

In the context of this work, two base classes from this module are used, the `QWebEngineView` and `QWebEnginePage`. These classes work together to render and manage web content in the browser.

- The `QWebEngineView` acts as a widget for displaying web pages and seamlessly integrates them into the application's user interface.
- `QWebEnginePage` is used to manage web page content, including navigation, JavaScript execution, and interaction with `HTML` elements.

QWebEngineView class

The `QWebEngineView` class is a key component that is used to display web pages or load web content. It acts as a container for web content and provides various navigation methods such as moving forward or backward in the browsing history, zooming in or out, and other typical interactions with web pages [16].

For seniors, the `QWebEngineView` class can be customized to increase usability and readability. This includes displaying web content and interfaces with larger text and simplified navigation controls, ensuring that the browser meets the accessibility needs of users. Adapting these features makes the browser more user-friendly, especially for senior users who may find standard web interfaces challenging.

QWebEnginePage class

The `QWebEnginePage` class, unlike the `QWebEngineView` class, manages the behavior of web content. It represents a single window of a web page and is responsible for executing JavaScript files and rendering web content. In addition, it provides various methods for managing the behavior of web elements and handling events related to the web page, such as clicking on links or downloading files [16].

This class is essential for enabling customization of default link behavior, which is critical for protecting seniors from fraudulent websites and other malicious activities on the Internet. By overriding these default behaviors, browsers can provide additional protective measures and customized features, thereby enhancing security and ensuring safer browsing for senior users.

4.3 HTTP Request and Response Handling Library

The Requests library is a powerful and user-friendly tool for managing `HTTP` communication. It simplifies the process of sending `HTTP` requests to a web server and processing the received responses. This abstraction removes the need to manually handle low-level operations such as compiling raw `HTTP` requests, managing headers, or encoding data [18].

The library supports a wide range of `HTTP` request methods, including:

- `GET`: Retrieve data from a specified resource.
- `POST`: Send data to create or update a resource.
- `PUT`: Update a resource.
- `DELETE`: Remove a specified resource.

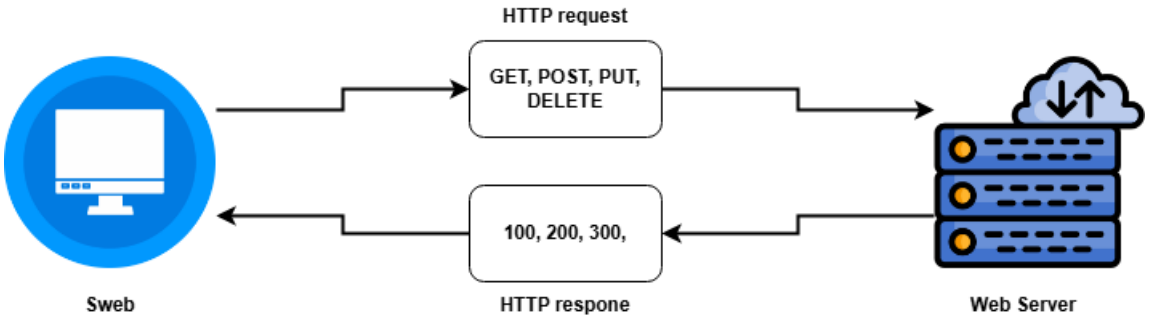


Fig. 4.5: Interaction between Sweb and Web Server via HTTP requests and responses.

In addition, the Request Library provides the following features:

- Automatic cookie processing to enable session management.
- Customizable headers that allow developers to add authentication tokens or user agent information.
- Timeout settings that provide the speed of response by determining how long the application waits for a server response.
- JSON data support, which makes it easy to work with modern APIs by converting JSON objects to Python dictionaries and vice versa.

Figure 4.5 demonstrates how the library can be used to implement an HTTP request, illustrating its straightforward syntax and practical application in real-world scenarios.

4.4 Library for File Interactions

When developing a web browser for seniors, it is essential to implement features to interact with different file types, including text files, JSON files, and TAR archives. These interactions are crucial for tasks such as managing user settings, saving data while browsing, and enabling data backup or transfer.

Json library

The JSON library is utilized for managing structured data. JSON (JavaScript Object Notation) is an ideal format for storing and retrieving configuration settings or user data in a lightweight and human-readable format [19]. In this application, JSON is used to store paths to text files and icons, as well as the default configuration for the browser. This includes settings such as default language, homepage, and toolbar customization tailored to senior users.

Tarfile library

The `tarfile` library is used to work with `TAR` archives [20], which can be used to create backups or restore saved browser configurations. `TAR` files allow the browser to combine multiple files, such as user data, settings, and saved sessions, into a single archive, making it easier for seniors to manage their data. The ability to compress and extract archives simplifies the process of transferring or securing important data.

4.5 System Libraries

The system library provides fundamental functions for interacting with the operating system and the user's device. In the context of this thesis, the following libraries are used:

- `os` Library
- `sys` Library
- `time` Library

Operating system library (os)

The os library is employed to perform operating system-level operations, such as file manipulation, directory traversal, and environment management. For instance, it is used to manage file paths for saving and retrieving user data [21].

In the context of this work, the library is used in a class designed to handle configuration data. This includes reading configuration settings stored in `JSON` files and also reading or writing specific data to text files. By leveraging the capabilities of the os library, the application provides efficient file management and seamless integration with the operating system.

System-specific parameters and functions library (sys)

The sys library serves as an important interface to the Python interpreter and provides access to various variables and functions in the application. One of its main roles is to provide an overview of the current state of the interpreter and to allow management of command line arguments via the `sys.argv` attribute. The `sys.argv` function is particularly useful when creating interactive command-line applications because it allows the program to parse and process user input directly from the command line [22].

In the context of this work, the `sys` library also supports the configuration of the application runtime environment, for example, dynamically modifying the execution path or exception handling.

Time library

The `time` library is essential for controlling and manipulating time in Python applications. It represents time in different formats, allows you to set delay intervals between events, and facilitates time conversion between different time formats. This feature is particularly important for managing time-based operations in an application, such as tracking session duration or scheduling tasks [23].

These precise timing features are used extensively in this work to ensure that the application responds quickly to user inputs, for example, when navigating between web pages or processing user-triggered events with minimal delay.

4.6 Machine Learning Libraries

To implement fraud detection mechanisms within the guardian application for seniors, several specialized libraries and modules are utilized. These tools provide the necessary infrastructure for building, training, and deploying machine learning models. This section outlines the key libraries employed in the system, highlighting their specific roles and contributions to the detection of fraudulent behavior.

Pandas library

`Pandas` is a powerful and widely used Python library for data manipulation and analysis. It provides efficient tools for handling structured data, particularly from tabular formats such as `CSV` files. Key functionalities of `Pandas` include data loading and saving, indexing, data cleaning, transformation, and basic statistical analysis. Additionally, it offers basic support for data visualization. Introduced in 2010 by Wes McKinney, Pandas has become a cornerstone library for data science, allowing for intuitive and efficient data handling in Python [30].

Scikit-learn library

`Scikit-learn`, often imported as `sklearn`, is a comprehensive library for machine learning in Python. It provides a broad selection of algorithms and tools for tasks such as classification, regression, clustering, dimensionality reduction, and model evaluation. The library also includes utilities for preprocessing data and performing validation techniques like cross-validation. Originally developed as a student project by David Cournapeau in 2007, `Scikit-learn` has evolved into a robust and user-friendly tool that is widely adopted in both academic research and practical applications. It is not included in the standard Python library and must be installed separately via package managers such as pip [32].

Joblib library

`Joblib` is a Python library used for the efficient serialization and deserialization of Python objects. In the context of machine learning, it is commonly employed to save and load trained models, especially those created using `Scikit-learn`. Compared to traditional methods like pickle, `Joblib` offers optimized performance for handling large numerical data, making it a practical choice when working with machine learning pipelines. Its simplicity and speed make it a valuable tool for model deployment and reuse [31].

4.7 Poetry

`Poetry` is a dependency management and packaging tool for Python that simplifies the process of creating, maintaining, and distributing Python projects. It handles everything from installing packages to managing virtual environments and defining project metadata in a clear and structured way using the `"pyproject.toml"` file [24]. Figure 4.6 illustrates the step-by-step process of how poetry install and poetry run work within a Python project, including the setup of dependencies and the execution of the application in an isolated environment.

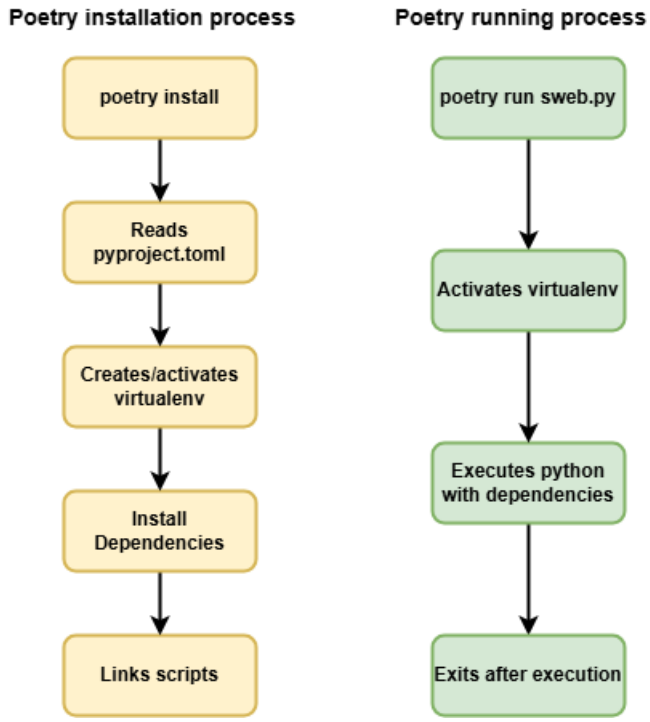


Fig. 4.6: Workflow of poetry install and `poetry` run in a Python Project [24].

In the context of the senior web browser (sweb) project, `Poetry` played a crucial role in streamlining the development workflow. All dependencies required for the application were explicitly declared and managed using Poetry, ensuring consistent environments across different systems and development stages. The listing 4.3 shows the required dependencies for the sweb.

Furthermore, Poetry was instrumental in the creation of the custom ISO image used to distribute sweb. By referencing the `"pyproject.toml"` file, the packaging process automatically included all necessary Python packages and dependencies, enabling the application to run smoothly in a live environment without the need for manual configuration. This approach not only saved time but also reduced potential compatibility issues, making deployment more reliable and repeatable.

Listing 4.3: Dependencies packages for sweb in the `pyproject.toml`.

```
[tool.poetry.dependencies]
python = "3.12"
pyqt5-qt5 = "5.15.2"
pyqt5 = "5.15.11"
pillow = "10.3.0"
pyqtwebengine-qt5 = "5.15.2"
pyqtwebengine = "5.15.7"
screeninfo = "0.8.1"
requests = "2.31.0"
dataclass-wizard = "0.26.0"
sconf = {path = "../sconf", develop = true }
joblib = "1.4.2"
scikit-learn = "1.6.1"
```

5 Design and Development of the Web Browser for Seniors

The chapter covers essential aspects of the browser's development, including the user interface design, phishing detection mechanism, multilingual support, and text readability enhancements. It also highlights the modular structure of the source code, which ensures maintainability and scalability. These topics are addressed through the following sections:
Section 5.1 outlines the conceptual foundation and goals of the senior web browser.
Section 5.2 details the configuration of the graphical development environment used to implement the browser.
Section 5.3 focuses on building an accessible and user-friendly interface tailored to senior users.

5.1 Conceptualizing the Senior Web Browser

As part of the operating system for the senior project, the web browser has been developed and adapted to the needs of senior users. This browser is one of several project suite applications, including an e-mail client, an app runner, and an operating system image. More information about the broader project is available in the GitHub repository https://github.com/forsenior/senior-os.

Figure 5.1 shows the project structure and highlights the web browser-specific folders and files. Configuration data for all applications is stored centrally in the sconf directory to ensure consistency across the entire suite of applications and simplify future development. This directory is organized into subdirectories containing various resources, such as:

- Icons displayed on application buttons.
- A database of fraudulent websites to protect users.

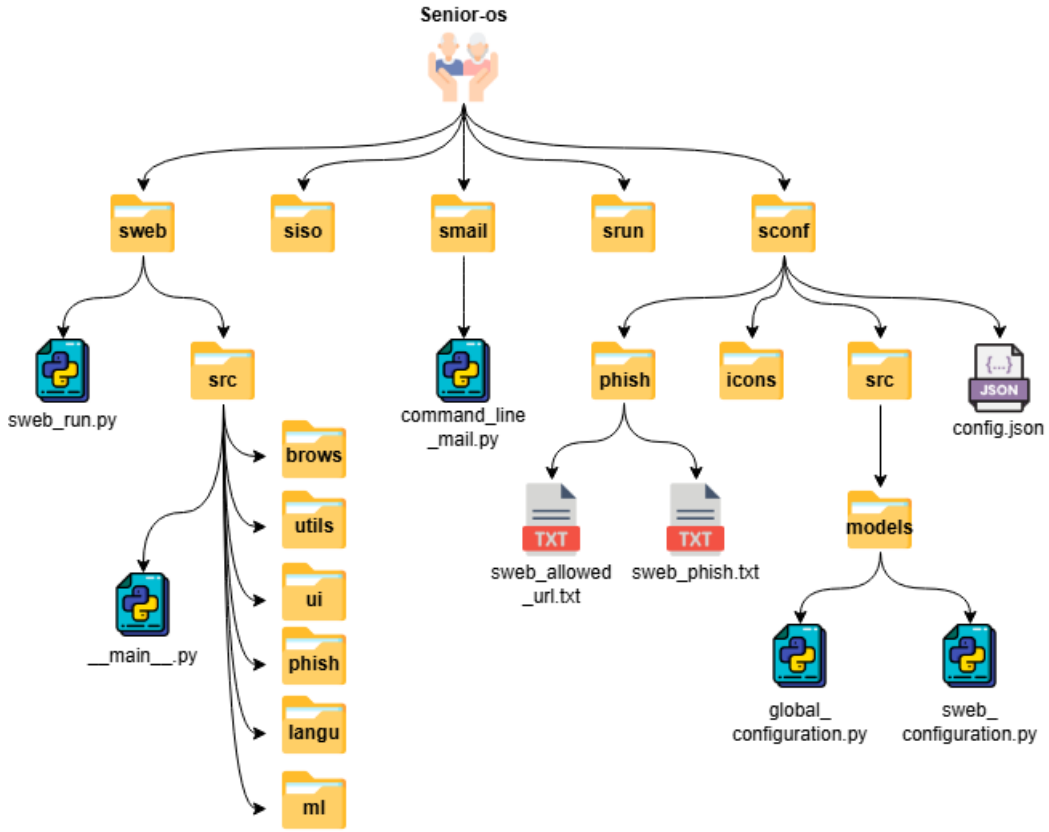


Fig. 5.1: Web browser directory overview for Senior-OS project.

Each application within the project has its own dedicated folder for programming and support files. In the case of the web browser 5.2, this folder contains:

- `Documents`: contains user manuals for the application, available in English (en), Czech (cz) and German (de).
- `Screenshots`: contains screenshots of the application that were used to create the readme.md file.
- src: Contains the modules for the application that contains the core code.
- `README.md`: Provides detailed instructions on how to install and run the application, along with general explanations.
- `requirements`: Specifies the library dependencies needed to run the web browser.
- `pyproject.toml`: Defines the project's metadata, dependencies, and configuration for Poetry-based packaging and environment management.
- `sweb_run.py`: The main script for launching the web browser.

This clear folder structure simplifies navigation, promotes clarity in project management, and ensures efficient development and maintenance of the web browser.

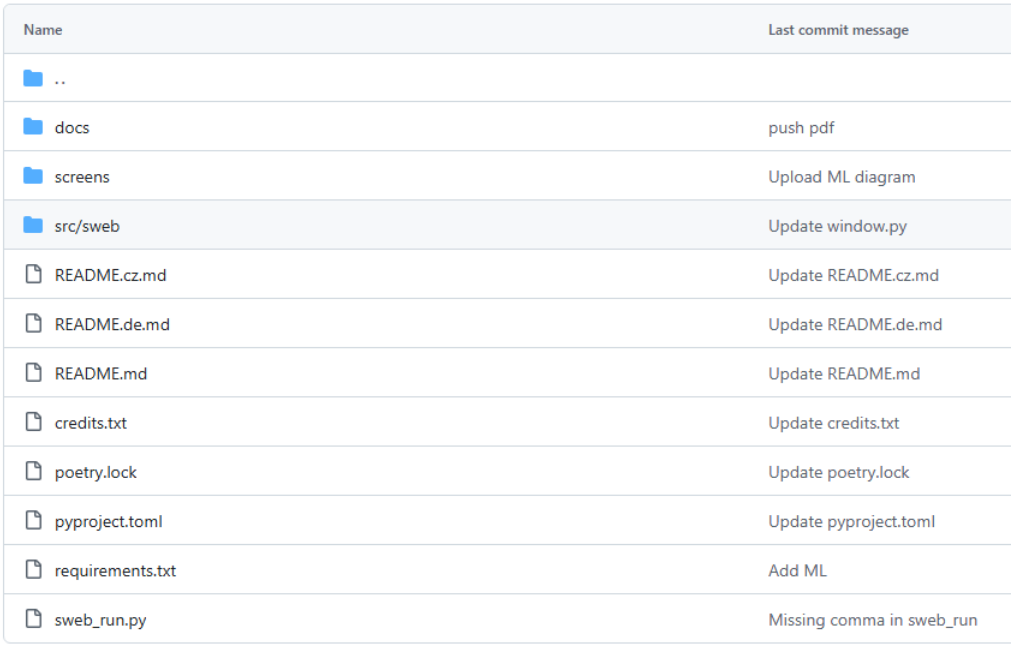


Fig. 5.2: Sweb application repository overview.

Details of the specific configuration data used for the web browser for seniors (sweb) are described in the following section. The senior web browser is designed with two main menus, but only one menu is displayed in the interface at any given time. Each menu contains five buttons, each of which is assigned a specific function. The layout and placement of these buttons are shown in Figure 5.3. In the middle of the browser interface is a dedicated area for displaying web content, which allows users to interact seamlessly.

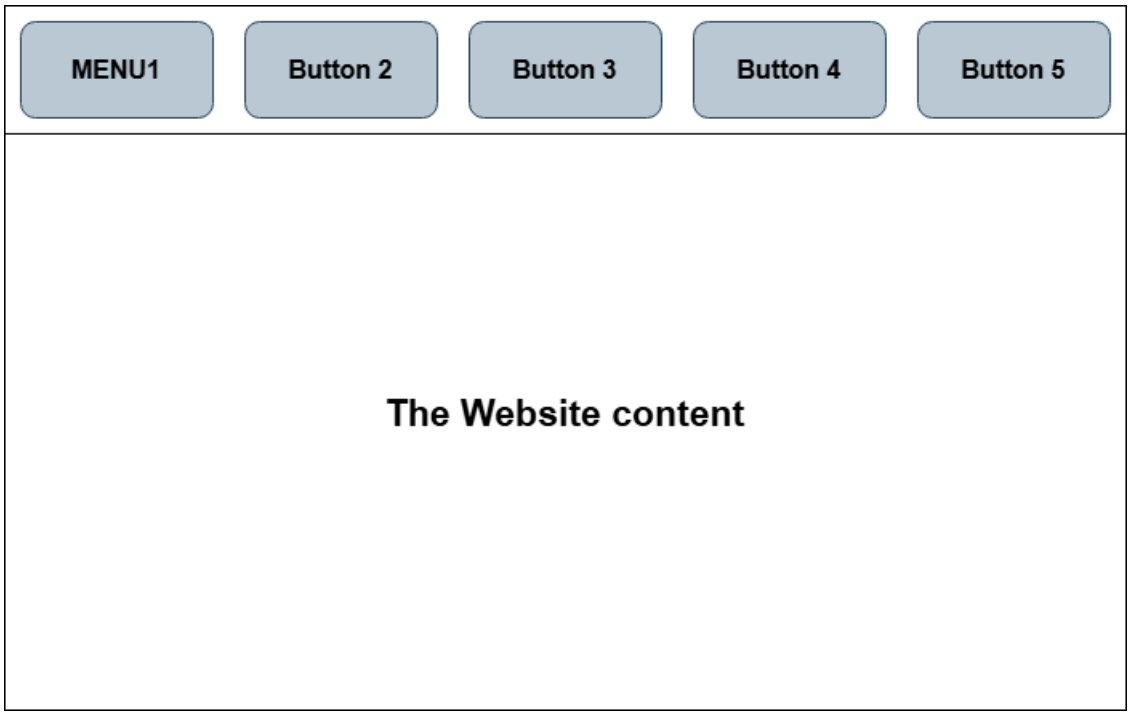


Fig. 5.3: Design of the first menu.

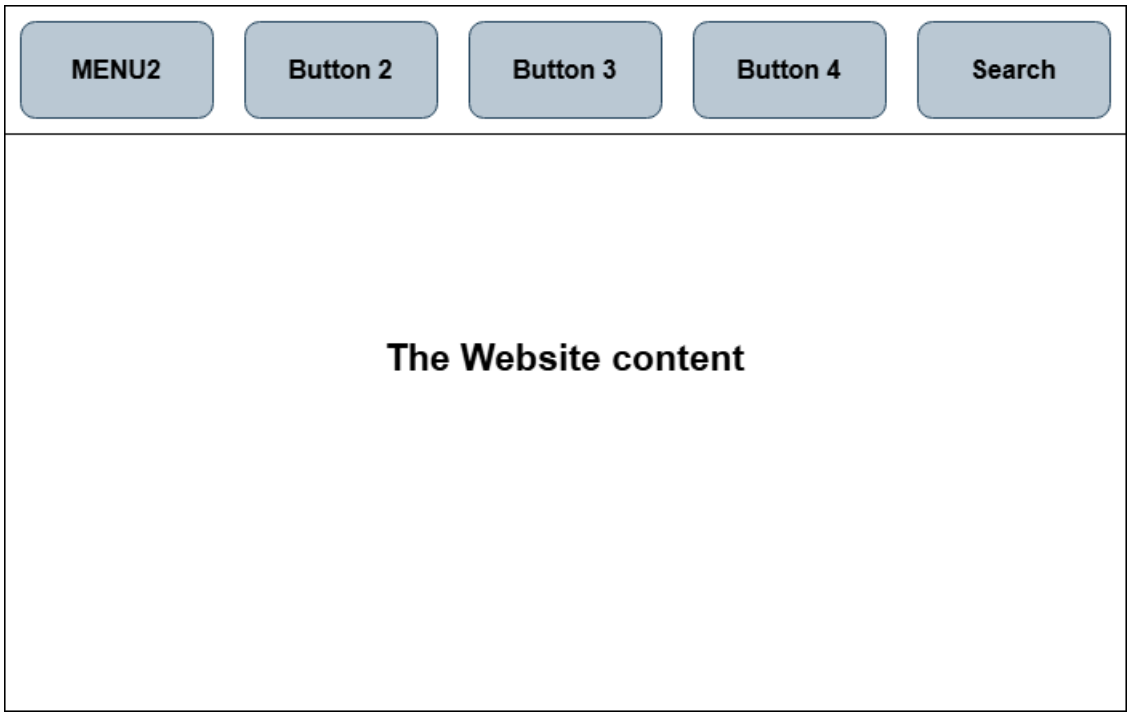


Fig. 5.4: Design of the second menu.

The two menus have a similar structure, with the main difference being the function of the last button in menu 2. This button, when clicked, opens an input

box in which users can enter a search query or the web address they wish to navigate to. If the user clicks the search button, the web content area temporarily disappears and the search box comes to the foreground, as shown in Figure 5.5.

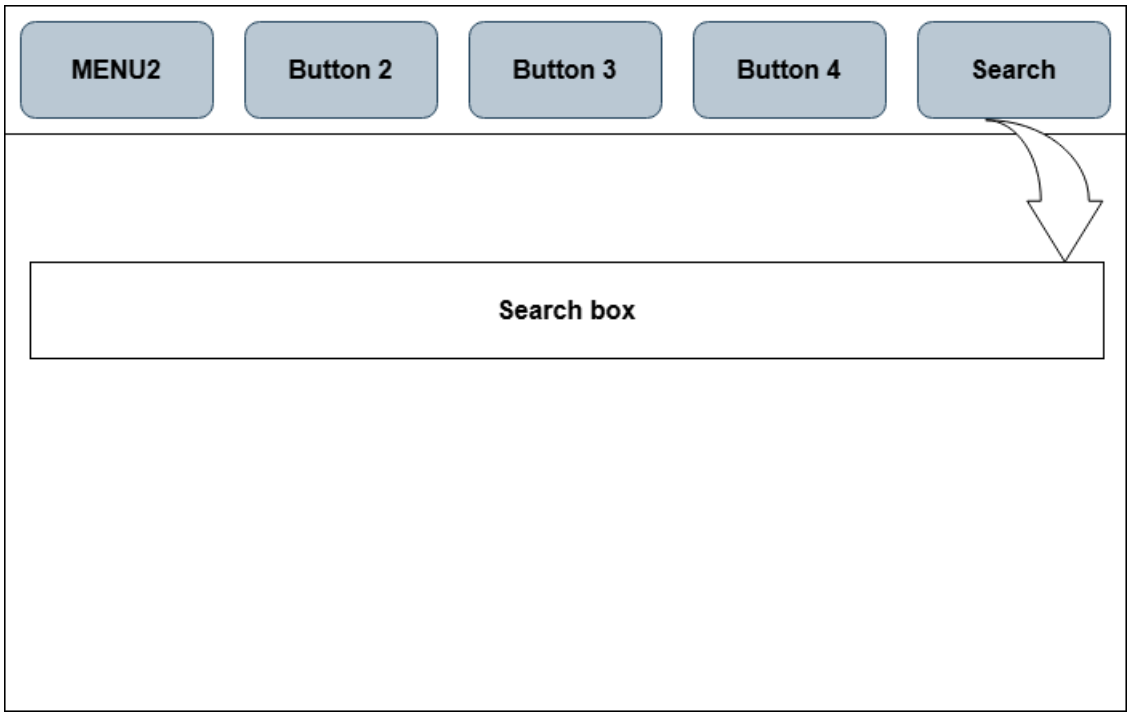


Fig. 5.5: Search box in web browser.

For seniors, the web browser emphasizes readability by supporting large text and icons. The size of text and icons in buttons can be pre-set in the sweb configuration file and the initialized values are used when creating buttons. However, increasing the text size for web content directly in `PyQt` poses a problem. This problem is solved by using dynamic `HTML` editing, as shown in 5.6

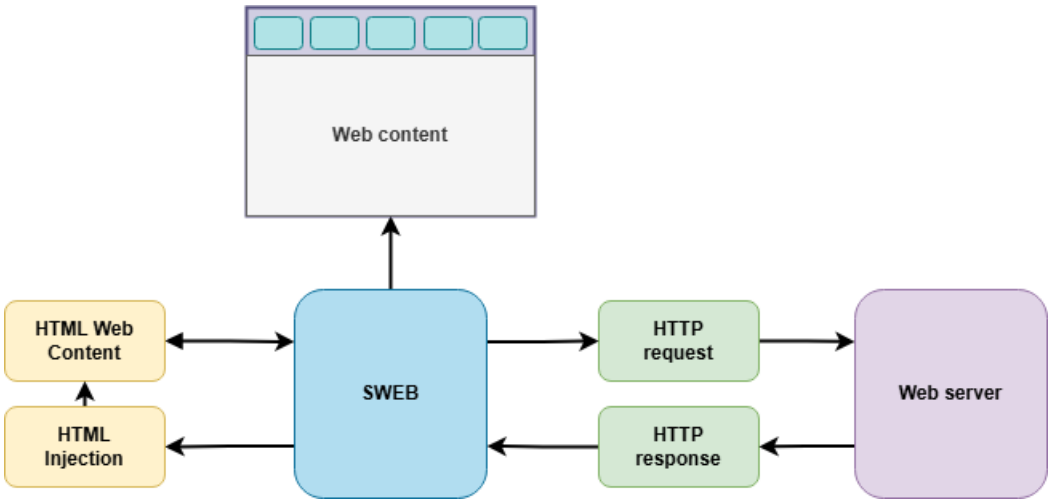


Fig. 5.6: Method of applying an HTML change to increase the size of text in web content.

The browser also includes security features that protect users from fraudulent web pages. For example, browser buttons turn red when users encounter potentially phishing sites, as demonstrated in Figure 5.7.

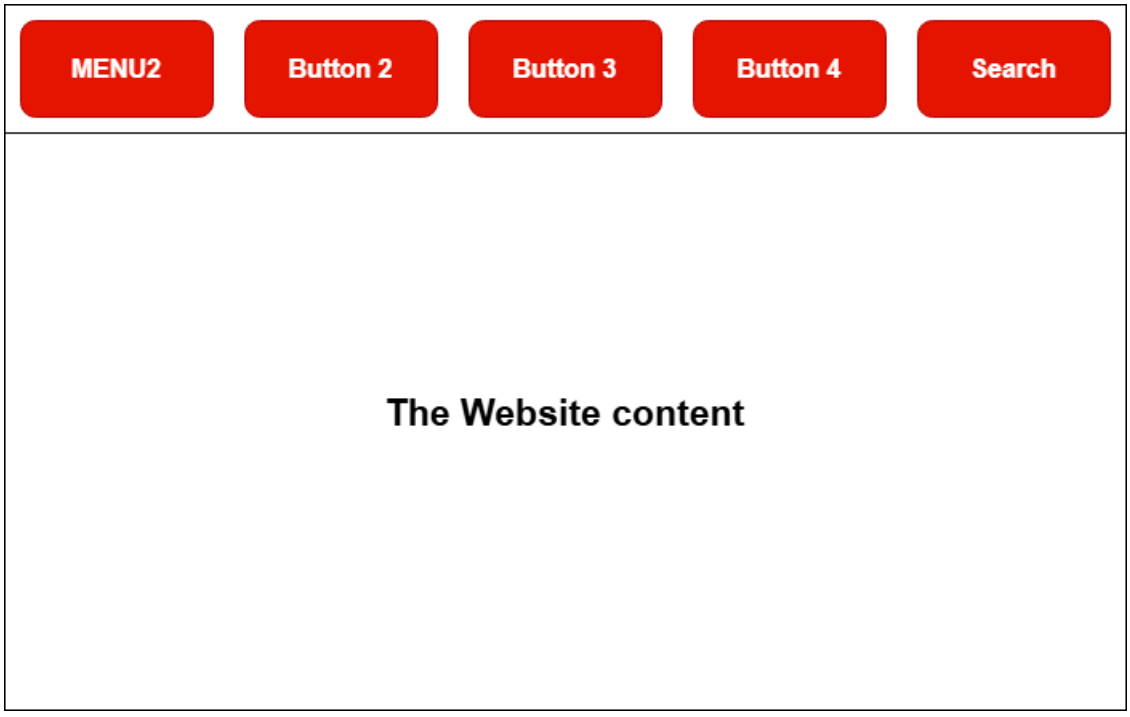


Fig. 5.7: Sweb interface when connecting to a phishing page.

A regularly updated database of scam pages is integral to maintaining user protection against phishing websites. The update process is automated and scheduled to occur every two weeks. The time elapsed since the last update is tracked to

ensure that the database remains current and effective. A suggestion for regularly updating the fraudulent page is shown in 5.8.

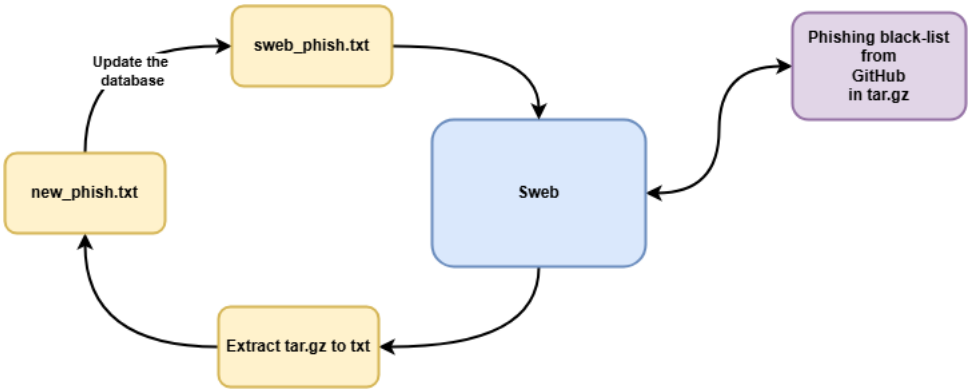


Fig. 5.8: Proposal to regularly update the blacklist of fraudulent websites.

In addition to the scam database, sweb includes integrating a neural network into the application as a second layer of protection. This neural network will analyze web pages in real-time to identify potential phishing sites based on patterns and behaviors that are difficult to detect through static databases alone. By combining a dynamic neural network approach with the existing scam page database, the app aims to provide robust and adaptive security for senior users. The mechanism by which URLs are blacklisted and evaluated by a machine learning model for phishing detection is shown in Figure 5.9.

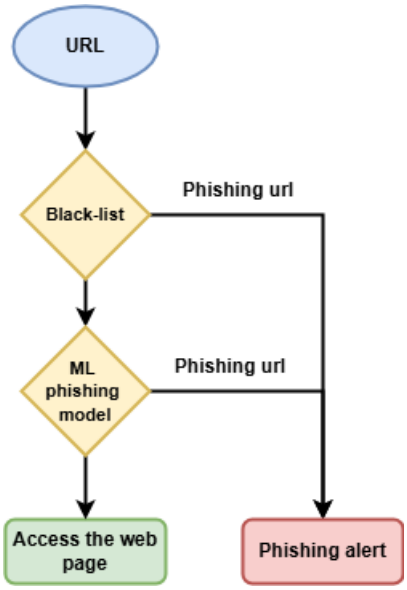


Fig. 5.9: Phishing detection mechanism in Sweb.

Additionally, the browser enforces a whitelist of allowed web pages, which is managed through the sconf application a component of the senior-os project. Users are not required to enter the full URL of a website; instead, entering just a portion of the web address is sufficient. The browser automatically matches this input to the first corresponding entry in the whitelist, simplifying access and reducing the need for users to remember complete URLs. This design enhances usability, especially for seniors, by making navigation more intuitive and less error-prone.

The use of the predefined list during internet browsing is governed by the selected protection level, which determines how strictly the application enforces access control. This mechanism will be explained in detail in the following chapter. Figure 5.10 visually illustrates the distinctions between the different protection levels. It is important to note that both the blacklist and the machine learning model are always activated when the user clicks on a sub-URL that falls under an allowed web page.

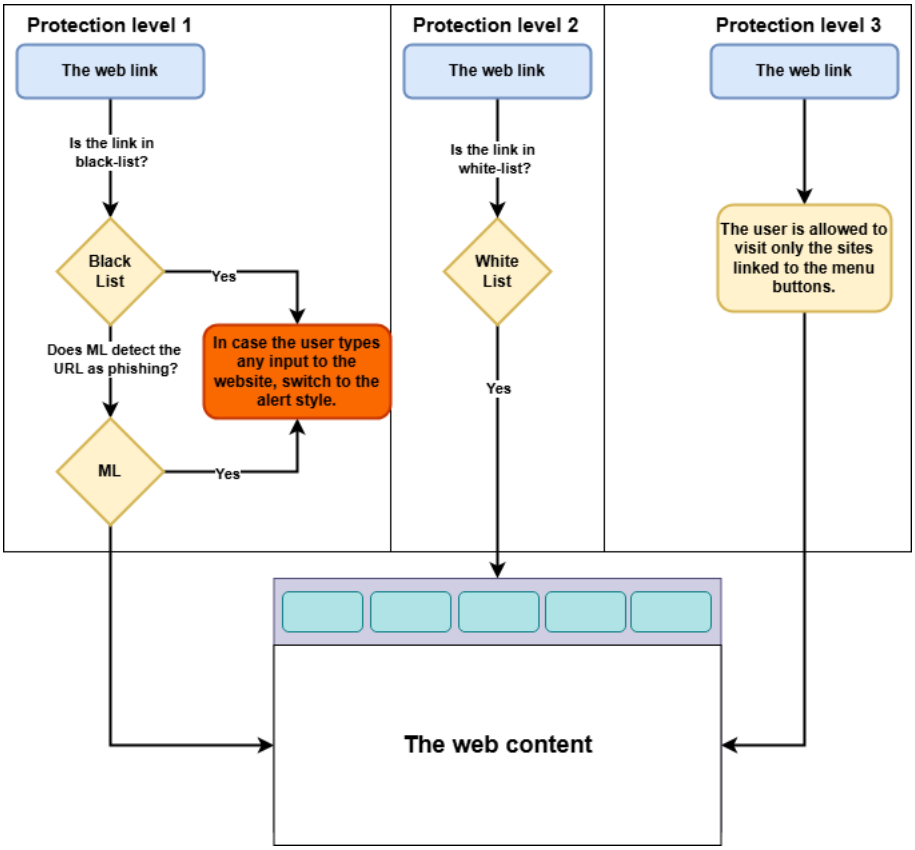


Fig. 5.10: Distinctions between the different protection levels.

5.2 Configuring the Graphical Development Environment

To begin, the main application must be initialized to manage essential tasks such as application startup, finalization, and cleanup when the application is terminated. This functionality is handled using the QApplication class, which is described in detail in Chapter 3.1 Graphics Library. A sample initialization sequence, including recalling configuration data, is illustrated in Listing 5.1.

Listing 5.1: View the application in full screen mode.

```
qApplication = QApplication(sys.argv)
# If browser is opened in command terminal
input_url_from_terminal = sys.argv[1] if len(sys.argv) > 1 else
    "https://vut.cz"
# Load config data from JSON file
main_window = MyBrowser(input_url_from_terminal,
_dataProvider.get_sweb_configuration(),
_dataProvider.get_global_configuration())
# Set parametr for main browser window
main_window.show()
# Call main browser window, this set the full screen.
main_window.show_app_full_screen()
sys.exit(qApplication.exec_())
```

The sweb supports two methods of initialization:

- **Command-Line Execution:** The browser can be launched directly using the python command. For example:

```
python sweb_run.py
```

- **Poetry-Based Execution:** If the project is managed using Poetry, the browser can be started with:

```
poetry run sweb
```

When executing the browser through the command line, a URL can be provided as an argument. This URL is assigned to the variable `input_url_from_terminal`. The script retrieves this argument using the `sys.argv` list, specifically its second element `(sys.argv[1])`, as the first element `(sys.argv[0])` represents the script name. A check ensures that the argument list contains more than one item to confirm that a URL was provided. If no URL is passed, the browser defaults to

opening https://vut.cz. Command-line execution is crucial for integration with the smail application. If the user clicks a link in the email client, smail can invoke the web browser (sweb) and display the desired webpage seamlessly.

Once initialized, the application retrieves its configuration settings through a designated provider known as the `dataProvider` method. This method is consistently used across all applications within the senior-os project to streamline configuration management. It allows sweb to automatically load required settings such as `sweb_configuration` and `global_configuration` without manually reading JSON files within the application logic.

The configuration data is organized into separate files to maintain modularity. For instance:

- `sweb_dataProvider` supplies configurations specific to the Sweb browser, including button icons, website links, paths to the blacklist and whitelist files, and more. The structure and content of `sweb_configuration` is shown in Listing 5.2.

Listing 5.2: Sweb configuration content.

```
swebAllowedUrlListV2: List[dict] = field(default_factory=lambda :
[
{"url1": "https://irozhlas.cz/", "icon1":
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/icons
/sweb_www_1.png"},
{"url2": "https://edition.cnn.com/", "icon2":
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/icons
/sweb_www_2.png"},
{"url3": "https://www.vut.cz/", "icon3":
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/icons
/sweb_www_3.png"},
{"url4": "https://www.info-zdravi.cz/", "icon4":
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/icons
/sweb_www_4.png"},
{"url5": "https://www.aktualne.cz/", "icon5":
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/icons
/sweb_www_5.png"},
{"url6": "https://www.denik.cz/", "icon6":
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/icons
/sweb_www_6.png"},
])

picturePaths: List[str] = field(default_factory=lambda:
```

```
    ["/run/archiso/airootfs/usr/lib/python3.13/site-packages/icons
/exit.png"])
allowedURL: str =
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/phish
/sweb_allowed_url.txt"
phishingDatabase: str =
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/phish
/sweb_phish.txt"
phishingDetectionModel: str =
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/sweb
/ml/phishing_detection_model.pkl"
phishingVectorizer: str =
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/sweb
/ml/tfidf_vectorizer.pkl"
command_line_mail_script: str =
    "/run/archiso/airootfs/usr/lib/python3.13/site-packages/smail
/connection/command_line_mail.py"
phishingGithubDatabase: str =
    "https://github.com/mitchellkrogza/Phishing.Database/raw/master
/ALL-phishing-domains.tar.gz/"
sendPhishingWarning: bool = True
...
...
```

- `global_dataProvider` delivers shared configuration data used across all senior-os applications, such as protection levels, default language, alert colors, and similar global settings. This is detailed in Listing 5.3.

Listing 5.3: Global configuration content.

```
    class GlobalConfiguration:
    language: str = "en"
    colorMode: str = "light"
    alertColor: str = "F90000"
    highlightColor: str = "48843F"
    protectionLevel: int = 1
    careGiverEmail: str = "test@gmail.com"
```

The mechanism of how the `dataProvider` functions is illustrated in Figure 5.11, while the integration of the provider into Sweb's codebase is demonstrated in Listing 5.4.

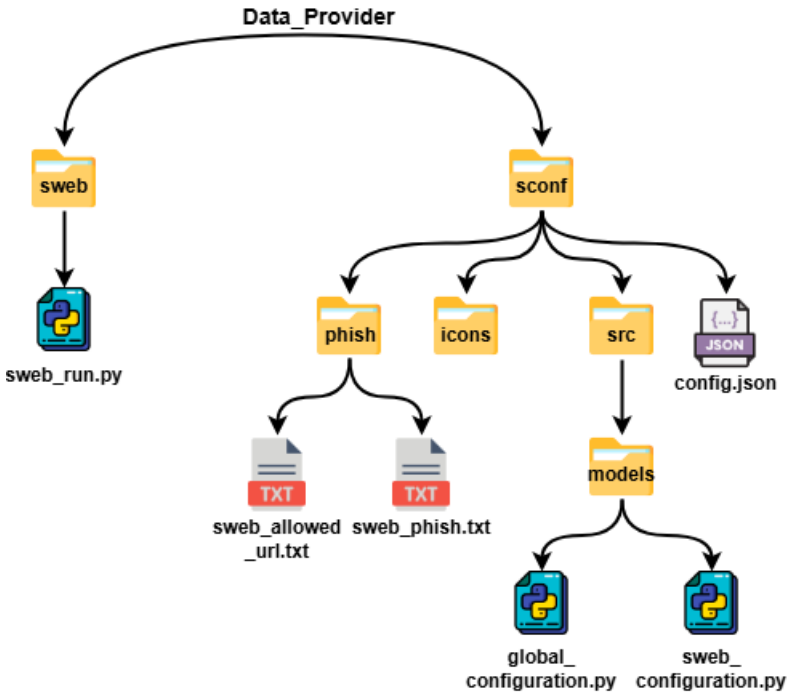


Fig. 5.11: The `_dataProvider` method.

Listing 5.4: Importing the `dataProvider` to sweb.

```
import sconf.configuration.configuration_provider as dataProvider
_dataProvider: dataProvider.ConfigurationProvider
def main():
   _dataProvider = dataProvider.ConfigurationProvider()
```

To ensure an optimal user experience, the main application window is displayed in full-screen mode without standard window decorations, such as headers or buttons for minimizing or closing. This is achieved through a two-step process:

- Window Flags Configuration: The window behavior is customized by setting the `CustomizeWindowHint` flag, which removes standard controls.
- Full-Screen Display: The `showFullScreen()` method is then used to present the main window in full-screen mode.

The corresponding commands for setting window flags and enabling full-screen mode are detailed in Listing 5.5. This streamlined approach ensures that the web browser interface remains uncluttered, enhancing usability for senior users.

Listing 5.5: View the application in full-screen mode.

```
   main_window.show()
   # Call main browser window, this set the full screen.
   main_window.show_app_full_screen()
```

Website view

The `QWebEngineView` class is central to displaying web content within the application, as illustrated in Listing 5.6. This class allows a web browser to be seamlessly embedded into a desktop application. The application itself is initialized using the `QApplication` class, providing the foundation for integrating standard web content into the browser interface.

Listing 5.6: Implementation the QWebEngine in the code.

```
self.main_browser = QWebEngineView()
self.ml_phishing_url_detector =
    PhishingURLDetector(sweb_dataProvider.phishingDetectionModel,
    sweb_dataProvider.phishingVectorizer)
# Set cutstom page to open new page in the same browser
self.my_custom_page = MyWebEnginePage(self.main_browser)
```

In some web pages, `hyperlinks` are set to open in new tabs or windows by default, typically using the `target="_blank"` attribute. However, this behavior is not ideal for the `QWebEngineView` class, which is designed to display all web content within a single window instance. To address this limitation, the default browser behavior needs to be overridden, as shown in Figure 5.12.

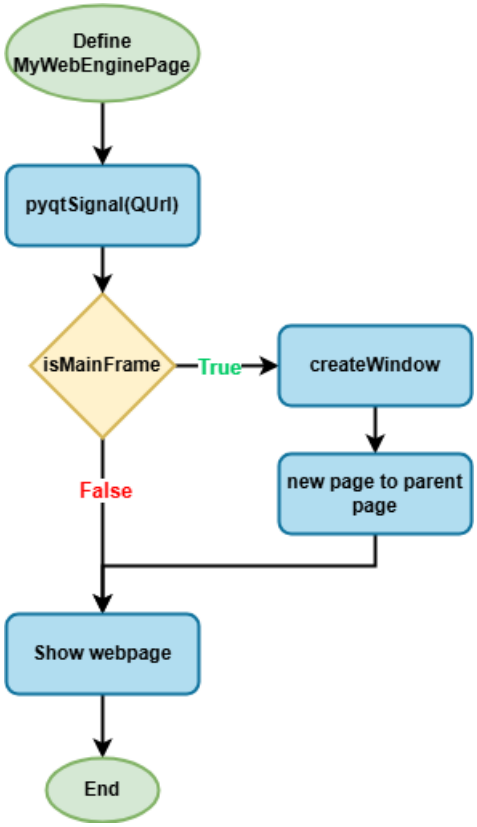


Fig. 5.12: Flowchart for assigning a new page to the current window.

To manage this effectively, a custom class called `MyWebEnginePage` is introduced, as shown in Listing 5.7. This custom class intercepts and handles navigation requests generated by the web content, particularly when links are clicked. By doing so, multiple instances of `QWebEngineView` are managed and displayed within the same window, ensuring a consistent user experience without spawning additional browser windows.

Listing 5.7: Customizing standard web content behavior.

```
class MyWebEnginePage(QWebEnginePage):

    # Define a signal that will carry a URL as its argument
    urlChangedSignal = pyqtSignal(QUrl)

    def __init__(self, parent=None):
        super().__init__(parent)
        # Create a channel for recording filling text when user fill text
            in phishing page
        self.channel = QWebChannel(self)
        self.setWebChannel(self.channel)
```

```
11
12      def acceptNavigationRequest(self, url, _type, isMainFrame):
13          # Ensure only modifying behavior for clicked links
14          if _type == QWebEnginePage.NavigationTypeLinkClicked and
                isMainFrame:
15              # Navigate to the url
16              self.urlChangedSignal.emit(url)
17              # Tell the view that handled this navigation request
18              return False
19          # Return True for all other navigation requests
20          return True
21
22      def createWindow(self, _type):
23          # Create a new instance of MyWebEnginePage for the new window
                request
24          new_page = MyWebEnginePage(self)
25          new_page.urlChangedSignal.connect(self.urlChangedSignal.emit)
26          return new_page
27
28      # Method for configuration in Mobile user agent
29      def setUserAgent(self, user_agent):
30          self.profile().setHttpUserAgent(user_agent)
```

5.3 Building an Accessible User Interface

The primary objective of this thesis is to design and implement a web browser specifically tailored for seniors. To achieve this, the application must prioritize accessibility and ease of use. This includes incorporating features such as large, clearly labeled buttons and icons, legible text, and support for multiple languages to accommodate a diverse user base.

The configuration of the graphical user interface (GUI) is defined in the `config.json` file, which is located in the `sconf` directory. This file contains structured data that controls the layout, appearance, and behavior of various elements in the Sweb browser. An excerpt of its content is shown in Listing 5.8.

Among the key components of this configuration are:

- `swebAllowedUrlListV2`: A list of dictionaries, each mapping a specific URL to an associated icon. These entries define the allowed websites accessible through the browser, each represented visually by a button with a corresponding icon. This list also determines the structure of the browser menus (e.g.,

two menus with four buttons each).

- `picturePaths`: Specifies the file path to images used in the browser, such as the exit icon.
- `allowedURL`: A path to a text file that contains the list of approved URLs for browsing, used to enforce the whitelist functionality.
- `phishingDatabase and phishingGithubDatabase`: Paths to static phishing databases used for detecting fraudulent websites, including a locally stored file and a remote backup hosted on GitHub.
- `phishingDetectionModel and phishingVectorizer`: Paths to machine learning model files used for real-time phishing detection. These files include a pre-trained classification model and its associated TF-IDF vectorizer.
- `command_line_mail_script`: Specifies the path to the script used for sending email alerts, which is triggered when sensitive input is detected on a suspicious webpage.
- `text`: A list of UI label strings in multiple languages (e.g., English, Czech, German) used for multilingual menu rendering.

This structured configuration approach allows the browser to be highly customizable without requiring changes to the underlying codebase. By externalizing these settings, developers and maintainers can easily update the browser's behavior, interface, and supported resources.

Listing 5.8: Configuration parameters for graphical user interface.

```
"swebConfiguration":
{"swebAllowedUrlListV2":
[{"url1": "https://irozhlas.cz/", "icon1":
    "../sconf/icons/sweb_www_1.png"},
{"url2": "https://edition.cnn.com/", "icon2":
    "../sconf/icons/sweb_www_2.png"},
{"url3": "https://www.vut.cz/", "icon3": "../sconf/icons/sweb_www_3.png"},
{"url4": "https://google.com/", "icon4": "../sconf/icons/sweb_www_4.jpg"},
{"url5": "https://www.aktualne.cz/", "icon5":
    "../sconf/icons/sweb_www_5.png"},
{"url6": "https://www.denik.cz/", "icon6":
    "../sconf/icons/sweb_www_6.png"}],

"urlsForWebsites": ["https://irozhlas.cz/", "https://edition.cnn.com/",
    "https://www.vut.cz/", "https://www.info-zdravi.cz/",
    "https://www.aktualne.cz/", "https://www.denik.cz/"],

```

```
"picturePaths": ["../sconf/icons/exit.png",
    "../sconf/icons/sweb_www_1.png", "../sconf/icons/sweb_www_2.png",
    "../sconf/icons/sweb_www_3.png", "../sconf/icons/sweb_www_4.png",
    "../sconf/icons/sweb_www_5.png", "../sconf/icons/sweb_www_6.png"],
"allowedURL": "../sconf/phish/sweb_allowed_url.txt",
"command_line_mail_script":
    "../smail/src/smail/connection/command_line_mail.py",
"phishingDatabase": "../sconf/phish/sweb_phish.txt",
"text": ["MENU 1", "MENU 2", "Search", "MENU 1", "MENU 2",
    "Vyhled\u00e1v\u00e1n\u00ed", "MENU 1", "MENU 2", "Suche"]},
```

To create buttons, the `QPushButton` class is utilized, as described in the previous chapter. A consistent design rule is applied to all buttons: each button can either display text or an icon, but not both simultaneously. Text is added to buttons using the `QLabel` class, while icons are added using the `QIcon` class. The layout of text or icons is managed using the `QVBoxLayout` class for straightforward alignment and control.

The clicked signal is employed to detect user interactions with buttons created using the `QPushButton` class, as demonstrated in Listing 5.9. This signal is emitted whenever a button is clicked. The connect method is then used to link this signal to a corresponding slot, triggering the desired functionality. Additionally, the cursor changes to a pointing hand `PointingHandCursor` when hovering over any button, achieved using the `setCursor` method from the `Qt` class.

Listing 5.9: Method for getting a click signal and changing the standard cursor.

```
menu1WWW1_layout.setAlignment(menu1WWW1_label,Qt.AlignCenter)
self.menu1WWW1.clicked.connect(self.navigate_www1)
self.menu1WWW1.setCursor(Qt.PointingHandCursor)
self.menu_1_toolbar.addWidget(self.menu1WWW1)
```

To add buttons to the application's menu interface, the `addWidget` method from the `QToolbar` class is used. For streamlined management, all buttons are grouped within a separate toolbar. The commands for assigning text or icons to buttons are provided in Listing 5.10.

Listing 5.10: Code to add text and icon to the button.

```
menu1WWW1_icon = QIcon(self.path_to_image_www1)
menu1WWW1_label = QLabel(self.menu1WWW1)
menu1WWW1_label.setPixmap(menu1WWW1_icon.pixmap(QSize(int
(self.buttons_width_info/(1.5)),int(self.buttons_height_info/(1.5)))))
menu1WWW1_layout.addWidget(menu1WWW1_label)
```

The structure and configuration of all menus and buttons within the application are detailed in Table 5.1.

Table 5.1: Controlling the application page.

Button Name	Content	Click Signal	Description
Menu 1	Text	Self.toggle_toolbar	Switching to menu 2
Menu 2	Text	Self.toggle_toolbar	Switching to menu 1
Exit	Icon	Self.close	Exit the application
WWW1	Icon	Self.navigate_www1	Redirect to website https://irozhlas.cz
WWW2	Icon	Self.navigate_www2	Redirect to website https://edition.cnn.com
WWW3	Icon	Self.navigate_www3	Redirect to website https://www.vut.cz
WWW4	Icon	Self.navigate_www4	Redirect to website https://www.info-zdravi.cz
WWW5	Icon	Self.navigate_www5	Redirect to website https://www.aktualne.cz
WWW6	Icon	Self.navigate_www6	Redirect to website https://www.denik.cz
Search	Text	Self.toggle_url_toobar	Display URL input field

Switching between two menus

The `toggle_between_toolbar` method is a custom function designed to seamlessly switch between Menu 1 and Menu 2 in the application's graphical user interface (GUI). This method, detailed in Listing 5.11, provides the functionality to toggle the visibility of the two menus.

Using the `self` parameter, the method references the web application's instance. By default, Menu 1 is displayed as the primary interface, while Menu 2 is initially hidden beneath Menu 1. The visibility of each menu is controlled using the `isVisible` method from the `QToolbar` class. This method accepts two values: True to make a menu visible and False to hide it.

Listing 5.11: Method for switching between two menus.

```
def toggle_between_toolbar(self):
    # Toggle visibility of toolbars
    if self.menu_1_toolbar.isVisible():
        self.menu_1_toolbar.setVisible(False)
        self.menu_2_toolbar.setVisible(True)
    else:
        self.menu_2_toolbar.setVisible(False)
        self.menu_1_toolbar.setVisible(True)
```

Whenever a menu click signal is received, the method determines which menu is currently active and switches the visibility status accordingly, ensuring a smooth transition between the two menus.

The `toggle_between_toolbar` method is linked to the menu buttons using the clicked signal and the connect method. The specific implementation of this linkage is shown in Listing 5.12.

Listing 5.12: Assigning the `toggle_between_toolbar` method to the button.

```
self.menu1_button.setCursor(Qt.PointingHandCursor)
self.menu1_button.clicked.connect(self.toggle_between_toolbar)
self.menu_1_toolbar.addWidget(self.menu1_button)
```

Create a URL input field

The URL input field is created using the `QLineEdit` class, which is ideal for single-line text inputs, such as entering web addresses or search queries. To enhance usability, the `setAlignment` method is used to center-align the text within the input field. The alignment is set to `AlignCenter` from the Qt class, ensuring that any text entered by the user appears neatly centered within the field. A custom style is also defined for the URL input field using the `setStyleSheet` method. This method allows customization of various visual properties, including colors, height, font families, font sizes, and more, as demonstrated in Listing 5.13. Additionally, the return Pressed signal of the `QLineEdit` class is connected to the custom `navigate_to_url` method. This signal is triggered whenever the user presses the Enter key while the `QLineEdit` field is in focus.

Listing 5.13: Create and set styles for the URL input text field.

```
# Create a URL bar
self.url_bar = QLineEdit()
self.url_bar.setAlignment(Qt.AlignCenter)
```

```
# Change the parameter of URL bar
self.url_bar.setStyleSheet(f"""
QToolBar {{
        background-color: {self.color_info_menu};
}}
QLineEdit {{
    border: 2px solid black;
    height: {self.buttons_height_info}px;
    font-family: 'Inter';
    font-size: {int(self.buttons_height_info/3)}px;
    font-weight: 'Regular';
    background-color: {self.color_info_app};
}}
""")
# When text of URL is changed, check for URL Phishing
self.url_bar.returnPressed.connect(self.navigate_to_url)
self.url_toolbar.addWidget(self.url_bar)
```

Customizing button properties

The button's height and width are statically defined in the code. Its properties are customized using the `setStyleSheet` method, which provides flexibility to adjust various aspects such as the button's color, hover effect, text size, font family, and other styling attributes. The text, created using the `QLabel` class, is also styled through this method. The default button set is shown in the listing 5.14.

Listing 5.14: Button's parameters in the default state.

```
def default_style_toolbar(self):
    if self.is_hex_color(self.global_dataProvider.highlightColor) !=
        False:
        HIGHLIGHTCOLOR =
            self.is_hex_color(self.global_dataProvider.highlightColor)
    else:
        HIGHLIGHTCOLOR = "#48843F"
    style_string = f"""
        QToolBar {{
        border: 0px solid transparent;
        background-color: transparent;
        spacing: 10px;
        }}
```

```
        QPushButton QLabel {{
            color: #FFFFFF;
        }}
        /* Changes parameters for button in Toolbar*/
        QPushButton {{
            border-radius: 3px;
            border: 1px solid #797979;
            background-color: #949494 ;
            margin: 20px 0px 20px 0px;
            font-size: 40px;
            font-weight: 'Regular';
            font-family: 'Inter';
            width: {BUTTON_WIDTH}px;
            height: {BUTTON_HEIGHT}px;
        }}
        QPushButton:hover {{
            background-color: {HIGHLIGHTCOLOR};
        }}
        QPushButton QLabel {{
            font-size: 40px;
            font-weight: 'Regular';
            font-family: 'Inter';
        }}
    """
    if Debug:
        print("The Default Style : ", style_string)
    return style_string
```

The phishing button style is shown in the listing 5.15. The method `is_hex_color` is used to verify whether the colors provided by `global_dataProvider.alertColor` and `global_dataProvider.highlightColor` are in valid HEX format. If the provided color value is not a valid HEX code, a default color `#F90000` is applied to ensure visual consistency and error handling.

Listing 5.15: Button's parameters in the phishing state.

```
def phishing_style_toolbar(self):
    if self.is_hex_color(self.global_dataProvider.alertColor)!= False:
        ALERCOLOR = self.is_hex_color(self.global_dataProvider.alertColor)
    else:
        ALERCOLOR = "#F90000"
    if self.is_hex_color(self.global_dataProvider.highlightColor)!= False:
```

```
        HIGHLIGHTCOLOR =
            self.is_hex_color(self.global_dataProvider.highlightColor)
    else:
        HIGHLIGHTCOLOR = "#48843F"
    alert_style_string = f"""
        QToolBar {{
        border: 0px solid transparent;
        background-color: transparent;
        spacing: 10px;
        }}
        QPushButton QLabel {{
            color: #FFFFFF;
        }}
        /* Changes parameters for button in Toolbar*/
        QPushButton {{
            border-radius: 3px;
            border: 1px solid #797979;
            background-color: {ALERCOLOR};
            margin: 20px 0px 20px 0px;
            font-size: 40px;
            font-weight: 'Regular';
            font-family: 'Inter';
            width: {BUTTON_WIDTH}px;
            height: {BUTTON_HEIGHT}px;
        }}
        QPushButton:hover {{
            background-color: {HIGHLIGHTCOLOR};
        }}
        QPushButton QLabel {{
            font-size: 40px;
            font-weight: 'Regular';
            font-family: 'Inter';
        }}
    """
    if Debug:
        print("The Alert Style: ", alert_style_string)

    return alert_style_string
```

The `setStyleSheet` method enables customization of various visual properties, such as the button's background color, the color displayed upon clicking, text size, and

font family, as well as the text itself, which is implemented using the `QLabel` class. The specific values set by this method are summarized in Table 5.2.

Table 5.2: Property and value set on the button.

Property	**Description**	**Value**
Border	Button border	`1xp solid #797979`
Margin	Margin	`20xp 0xp 20xp 0xp`
Font size	Font size	`40px`
Font weight	Font weight	`Regular`
Font family	Font family	`Inter`
Width	Button width	Value of `BUTTON_WIDTH`
Height	Button height	Value of `BUTTON_HEIGHT`
Background color	Button color when not selected	`#949494`
Hover color	Button color when selected	Value of `HIGHLIGHTCOLOR`

Multi-language application support

The `Translator` custom class is specifically designed to manage language translations, providing an efficient way to implement multi-language support within the web application. A sample implementation of the `Translator` class is shown in Listing 5.16.

Listing 5.16: Initializing the `Translator` class.

```
class Translator:
    # Create initial function
    def __init__(self, _dataProvider, global_dataProvider):
        # Default shortcut for language
        self.language_keys = ["cz","en","de"]
        self.current_language_in_browser = global_dataProvider.language
        self.current_language_in_browser =
            self.current_language_in_browser.lower()
        self.protection_level = global_dataProvider.protectionLevel
        # If language is not supported, set it to EN
        if self.current_language_in_browser not in self.language_keys:
```

```
            self.current_language_in_browser = "en"
        if Debug:
            print("The language is set to:
                ",self.current_language_in_browser)
        self.text = _dataProvider.text
        # Set current language is CZ =0 , EN = 1, DE = 2
        if(self.current_language_in_browser) == "cz":
            self.current_language_index = 0
        elif self.current_language_in_browser == "en":
            self.current_language_index = 1
        else:
            self.current_language_index = 2
    # Toggle between each languages
    def toggle_supported_language(self):
        # Increase the index to change the language
        self.current_language_index += 1
        # If we reached the end of the list, go back
        if self.current_language_index >= len(self.language_keys):
            self.current_language_index = 0
        # Change current language after this function called
        self.current_language_in_browser =
            self.language_keys[self.current_language_index]
    def get_translated_text(self, button_name):
        key_text = f"sweb_{self.current_language_in_browser}_{button_name}"
        if key_text == "sweb_en_menu1":
            return self.text[0]
        if key_text == "sweb_en_menu2":
            return self.text[1]
        if key_text == "sweb_en_menu2Address":
            if self.protection_level == 3:
                return self.text[3]
            return self.text[2]
        ...
        ...

    # Get state of current language
    def get_current_language(self):
        return self.language_keys[self.current_language_index]
```

This class handles several key functions, including loading language configuration data, switching between supported languages, and retrieving translation text from

the configuration file. For this thesis, the web browser currently supports three languages: Czech, English, and German. The default language is initialized as defined in the `config.json` file. During the initialization process (`__init__`) constructor, a list of language codes is created, where `"cz"` represents Czech, `"en"` represents English, and `"de"` represents German. A language index is used to keep track of the current language and facilitates seamless switching between the available options.

The `toggle_language` method allows users to switch to the next language in the list by incrementing the language index. This functionality ensures a user-friendly and accessible multi-language experience.

Listing 5.17: Multilingual button labels defined in `config.json`.

```
"text":
["MENU 1","MENU 2","Search","Disabled",
"MENU 1","MENU 2","Vyhledávání","Vypnuté",
"MENU 1","MENU 2","Suche","Deaktiviert"]
```

The multilingual support in the application is handled through text entries stored in the `config.json` file, located in the `sconf` directory in the project and located in `/home/$USER/` in the live ISO. Each button's label is updated using the `get_translated_text` method. This method constructs a key by combining the abbreviation of the current language retrieved from the `current_language` parameter with the name of the corresponding button. The resulting key follows the format `sweb_{self.current_language_in_browser}_{button_name}`. This key is then used to fetch the appropriate localized text value from the configuration file. For instance, the string associated with the key "text" is retrieved and returned as the translated label. If the protection level is set to 3 (indicating that the search functionality is disabled), the label of the 'Search' button is automatically set to 'Disabled'. Listing 5.17 shows an example of multilingual entries stored in the `config.json`, while Table 5.3 presents the translated text values used for different languages.

Table 5.3: Mapping of button text to protection levels.

Protection Level	**Button**	**English**	**Czech**	**German**
	Menu1	MENU 1	MENU 1	MENU 1
1	Menu2	MENU 2	MENU 2	MENU 2
2	Search	Search	Vyhledávání	Suche
3	Search	Disabled	Vypnuté	Deaktiviert

Enhancing Web Page Text Readability

Improving font size and readability on a web page is crucial for older users. However, due to the encapsulated nature of web content and differences between browser and webpage styles, directly modifying a webpage's style from the browser interface is not feasible. To address this, two primary methods are employed in this thesis:

- Altering the HTML content
- Adjusting the zoom factor.

Modifying HTML Content

For the first approach, HTML content modification, custom `JavaScript` code is injected into the currently displayed web page. This code dynamically alters the appearance and interaction behavior of selected elements within the page. By targeting specific `HTML` tags, the script is capable of adjusting the font sizes and layout properties of various elements, thereby enhancing text visibility and user experience. An example of such `HTML` content modification for increasing text size is shown in Listing 5.18.

Listing 5.18: Custom HTML content modification to increase the text size of a web page.

```
def html_injection_to_web_content(self):
    injection_javasript = """
    <!-- Change only paragraph, article, span and header elements with
        lower levels-->
    var all_changed_content_tag = ['p', 'div', 'article', 'span', 'h3',
        'h4', 'h5'];
    <!-- Create a function to change content style-->
    var change_content_style = function(element) {
        <!-- Method includes will return value in UPPERCASE>
        if (['H3', 'H4', 'H5', 'A'].includes(element.tagName)) {
            <!-- Header with bigger size-->
            element.style.fontSize = '16px';
            element.style.lineHeight = '1.0';
        }
        <!-- Method includes will return value in UPPERCASE>
        else if (['P', 'DIV', 'ARTICLE',
            'SPAN'].includes(element.tagName)) {
            <!-- Content with smaller size>
            element.style.fontSize = '12px';
            element.style.lineHeight = '1.1';
        }
```

```
19          Array.from(element.children).forEach(change_content_style);
20      }
21      change_content_style(document.body);
22      """
23      self.main_browser.page().runJavaScript(injection_javasript)
```

At the core of this method lies the `change_content_style` function. This function traverses the Document Object Model `(DOM)`, beginning with `document.body`, and recursively updates the styling of nested elements. By doing so, it ensures that modifications are consistently applied across the entire content area of the page.

The function distinguishes between header elements (e.g., `"H3", "H4", "H5", "A"`) and body text elements (e.g., `"P", "DIV", "ARTICLE", "SPAN"`). Header tags are styled with an enlarged font size of `20px` and a line height of `1.0`, whereas body text elements are set to `17px` with a line height of `1.1`. This separation allows for better typographical balance and significantly improves readability. The detailed workflow is illustrated in Figure 5.13.

To ensure comprehensive application of these changes, the function uses `Array.from(element.children).forEach(change_content_style)`, which converts the children collection of each `DOM` node into an iterable array and recursively applies the style changes to all descendant elements.

Once the `JavaScript` code is defined, it is executed via the `runJavaScript` method of the `QWebEnginePage` object, which represents the active web page loaded in sweb. This is initiated by calling the `page()` method on the `main_browser` object. The `JavaScript` code is stored in the `injection_javascript` variable and is executed asynchronously to dynamically modify the structure and appearance of the currently displayed web content. This implementation is demonstrated in the last line in 5.18.

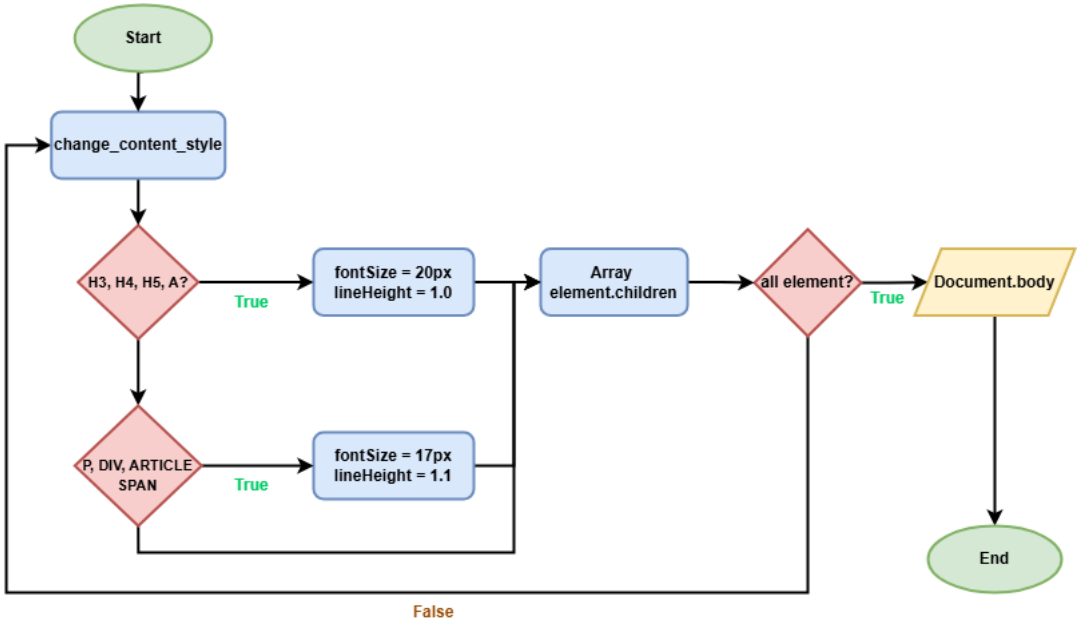


Fig. 5.13: Changing HTML content to increase text size.

Adjusting the Zoom Factor

The `setZoomFactor` method is employed to enhance the display size of the web page content. This method provides a powerful means of controlling the zoom level within the embedded browser. By setting a zoom factor greater than `1.0`, all visual elements on the page, such as text, images, and interface components—are proportionally enlarged. This feature is particularly beneficial for senios who may struggle with small text or densely packed layouts.

In this implementation, the zoom factor is set to `1.1`, which results in all elements being scaled to `110%` of their original size. This adjustment not only improves text readability but also enhances the visibility of images and interactive elements. The method is applied using the `main_browser` instance, which represents the primary browser component within the application. The corresponding implementation is illustrated in Listing 5.19.

Listing 5.19: Adjusting web page zoom level using `setZoomFactor`.

```
def finished_load_web_page(self):
    # Get url value from browser
    url_in_browser_value = self.main_browser.url().toString()
    senior_website_posting_option =
        self.sweb_dataProvider.seniorWebsitePosting
    permitted_website_list = self.permitted_website_list
```

```
    check_result = any(permitted_website in url_in_browser_value for
        permitted_website in permitted_website_list)
    if check_result:
        if "homepage.html" not in url_in_browser_value:
            self.main_browser.setZoomFactor(1.1)
            # Wait 1 second for loading, after 1 second, connect to change
                web content (HTML injection)
            QTimer.singleShot(250, lambda:
                self.html_injection_to_web_content())
    elif self.toggle_phishing_webpage:
        self.main_browser.setZoomFactor(1.1)
        # Wait 1 second for loading, after 1 second, connect to change web
            content (HTML injection)
        QTimer.singleShot(250, lambda:
            self.html_injection_to_phishing_web_content())
    else:

        if senior_website_posting_option:
            self.main_browser.setZoomFactor(1.1)
            # Wait 1 second for loading, after 1 second, connect to change
                web content (HTML injection)
            QTimer.singleShot(250, lambda:
                self.html_injection_to_web_content_strict())
        else:
            return
```

In conclusion, directly modifying the styles of web pages from within a PyQt based browser such as sweb poses certain limitations. Therefore, employing techniques like HTML content manipulation and zoom factor adjustment provides practical and effective solutions for enhancing readability. These methods are tailored to the specific needs of elderly users and contribute significantly to a more accessible and user-friendly browsing experience.

6 Enhancing Security in the Sweb Application

Web browsers tailored for seniors must prioritize security mechanisms to protect users from online threats, particularly fraudulent and phishing websites. In this thesis, the Sweb application integrates several security features designed to detect and alert users when they encounter potentially harmful web pages.

The browser employs a dual-layered protection system. First, it checks each visited URL against a locally maintained fraudulent site database, alerting the user and modifying the interface for example, by changing button colors to a predefined alert color (specified in the global configuration) whenever a match is detected. Second, it incorporates a lightweight machine learning model capable of evaluating the URL in real-time to identify phishing attempts based on patterns in the address.

To ensure ongoing protection, the fraudulent website database is updated automatically every two weeks. However, retraining the machine learning model is not part of this implementation, as the browser is designed to run from a Live ISO environment, where intensive operations such as model training could degrade system performance and produce unreliable results.

These functionalities are handled by a set of custom classes, which will be explained in the sections that follow.

Section 6.1 describes the core mechanism used to detect fraudulent websites.

Section 6.2 explains how the browser updates the fraudulent site database.

Section 6.3 discusses the training and integration of the machine learning model.

6.1 Securing Web Access Through URL Blacklist Verification

This subsection outlines the mechanism employed by the sweb browser to detect and protect users from accessing potentially fraudulent websites through URL blacklist verification.

The interception of URL changes is handled via the `urlChanged` signal provided by the web browser engine. As demonstrated in Listing 6.1, the following connection is established:

Listing 6.1: URL change capture.

```
1 self.main_browser.urlChanged.connect(self.security_against_phishing)
```

This line of code connects the `urlChanged` signal to the `security_against_phishing` method. Whenever the browser's active URL changes due to user navigation, link clicks, or search inputs, the signal is triggered, and the connected slot (i.e., `security_against_phishing`) is executed automatically.

Listing 6.2: Checking the web URL.

```
def security_against_phishing(self,qurl):
    # Get url from QURL
    url_in_browser_value = qurl.toString()
    if Debug:
        print("URL in Browser: ", url_in_browser_value)
    if url_in_browser_value.endswith('/'):
        if self.url_blocker.is_url_blocked(url_in_browser_value):
            self.toggle_phishing_webpage = True
            self.main_browser.setUrl(QUrl(url_in_browser_value))
            ...
            ...
        else:
            if url_in_browser_value.startswith("https://www.") or
                url_in_browser_value.startswith("http://www."):
            ...
            ...
    elif not url_in_browser_value.endswith('/'):
        if "about:blank" in url_in_browser_value:
            self.toggle_phishing_webpage = False
            return
        elif "google.com" in url_in_browser_value:
            self.toggle_phishing_webpage = False
            self.menu_1_toolbar.setStyleSheet(self.default_style_toolbar())
            self.menu_2_toolbar.setStyleSheet(self.default_style_toolbar())
            # Connect to URL after entering
            self.main_browser.setUrl(QUrl(url_in_browser_value))
        elif self.url_blocker.is_url_blocked(url_in_browser_value):
            self.toggle_phishing_webpage = True
            print("The third BL block condition is met")
            self.main_browser.setUrl(QUrl(url_in_browser_value))
            ...
            ...
        else:
            if url_in_browser_value.startswith("https://www.") or
                url_in_browser_value.startswith("http://www."):
```

```
34            ...
35            ...
36    self.main_browser.loadFinished.connect(self.finished_load_web_page)
```

The `security_against_phishing` method, which is listed in 6.2 is designed to inspect the new URL and determine whether it should be flagged as fraudulent. The process begins by filtering out irrelevant pages such as `"about:blank"`, which are considered safe placeholders and are therefore ignored to avoid unnecessary checks or logging.

If the URL is valid, the method continues by applying simple rule-based checks. For instance, URLs containing `"google.com"` are assumed to be trusted and bypass the fraud detection routine, although they are still logged in the activity history for traceability.

Next, the method calls the `url_blocked` function from the `URLBlocker` class to determine whether the current URL appears in a blacklist of known fraudulent domains.

The `URLBlocker` class plays a central role in blacklist-based detection. It maintains a set of blocked URLs loaded from the `sweb_phish.txt` file located in the `sconf/phish` directory. On live ISO systems, this file is stored in the runtime path: `"/run/archiso/airootfs/usr/lib/python3.13/site-packages/phish/sweb_phish.txt"`.

Listing 6.3: Managing fraudulent sites with a blacklist.

```
1  def load_urls_from_phishing_database(self,path):
2      try:
3          with open(path,"r") as openfile:
4              # Real all member from the file until the file is None
5                  if openfile is not None:
6                      content = openfile.read()
7                      reading_url = content.strip().split('\n')
8                      self.blocked_urls_database.update(reading_url)
9      except FileNotFoundError:
10         print("Phishing database not found, The application will continue
               to run without blocking any URLs")
11         pass
12 ...
13 ...
14
15 # Control that if the url is Block
16 def is_url_blocked(self, input_url):
```

```
    if "login.microsoft" in input_url:
        return False
    elif "microsoft365.com" in input_url:
        return False
    else:
        for blocked_url in self.blocked_urls_database:
            if blocked_url in input_url:
                return True
        return False
```

As illustrated in Listing 6.3, the class initializes an empty set to store blacklisted URLs. These URLs are populated by the `load_url_from_phishing_database` method, which reads the contents of the database file `sweb_phish.txt` whose path is provided by the `sweb_configuration_provider` through the `config.json`. Each line is stripped of whitespace and treated as an individual fraudulent URL. The set is then updated using the update method, ensuring fast lookup and efficient matching.

An overview of this mechanism is visually depicted in Figure 6.1, which illustrates the flow of URL monitoring and blacklist verification within the browser.

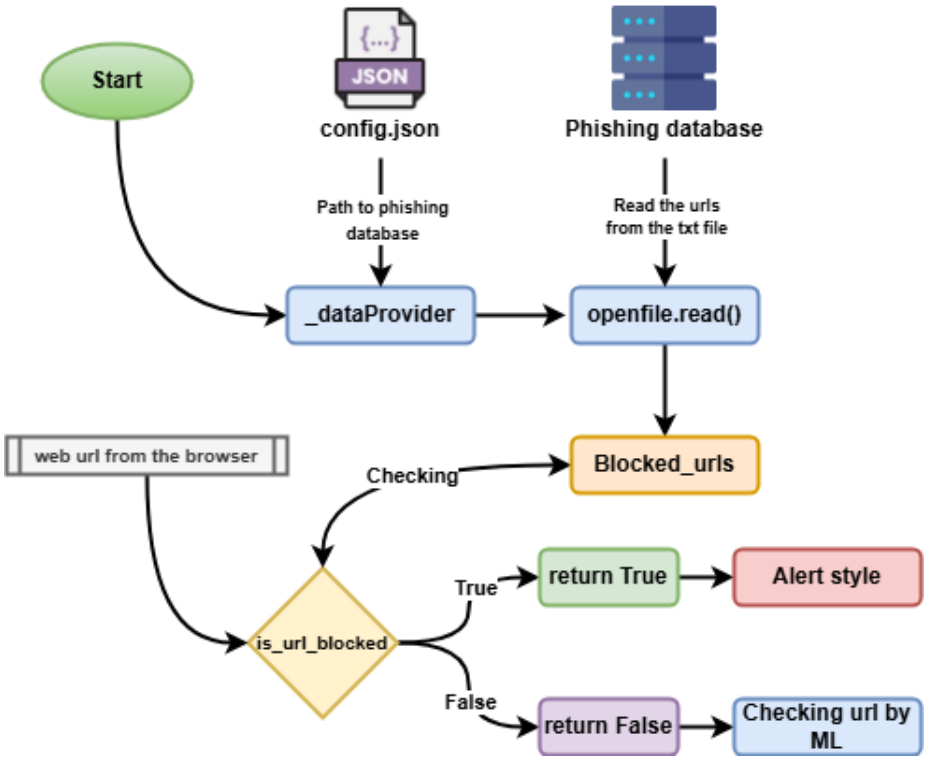


Fig. 6.1: URL verification based on blacklist within the sweb.

The `is_url_blocked` method is responsible for determining whether a specific URL should be classified as a phishing site. It accepts a single input parameter, url,

which is passed from the browser via the `urlChanged` signal. The method iterates through each entry in the phishing URL list and checks whether any of the listed URLs appear as substrings within the provided URL. If an exact or partial match is found, the method returns True, indicating that access to the URL should be blocked. Otherwise, it returns False.

In essence, the `URLBlocker` class encapsulates the logic for managing and verifying phishing URLs. It loads the list of known malicious domains from the `sweb_phish.txt` database and provides a streamlined method for checking whether a visited URL matches any entry in the list. This approach is particularly effective for enhancing web browsing security for elderly users. By using a substring matching technique, the method allows for flexible and broad detection of phishing attempts, contributing to a safer browsing environment.

6.2 Fraudulent URL List Synchronization

Regularly updating the fraudulent site database is critical to maintaining effective protection within the web browser. This is especially important given that phishing threats are constantly evolving, with attackers frequently creating new fraudulent websites and adopting increasingly sophisticated evasion techniques. Without routine updates, even a well-secured browser risks falling behind these fast-moving threats. By ensuring that the database consistently reflects the most recent known phishing URLs, users benefit from improved security and reduced exposure to online scams. This update process is handled through two custom classes: `PhishingDatabaseModificationChecker` and `FileUpdater`, illustrated in Figure 6.2.

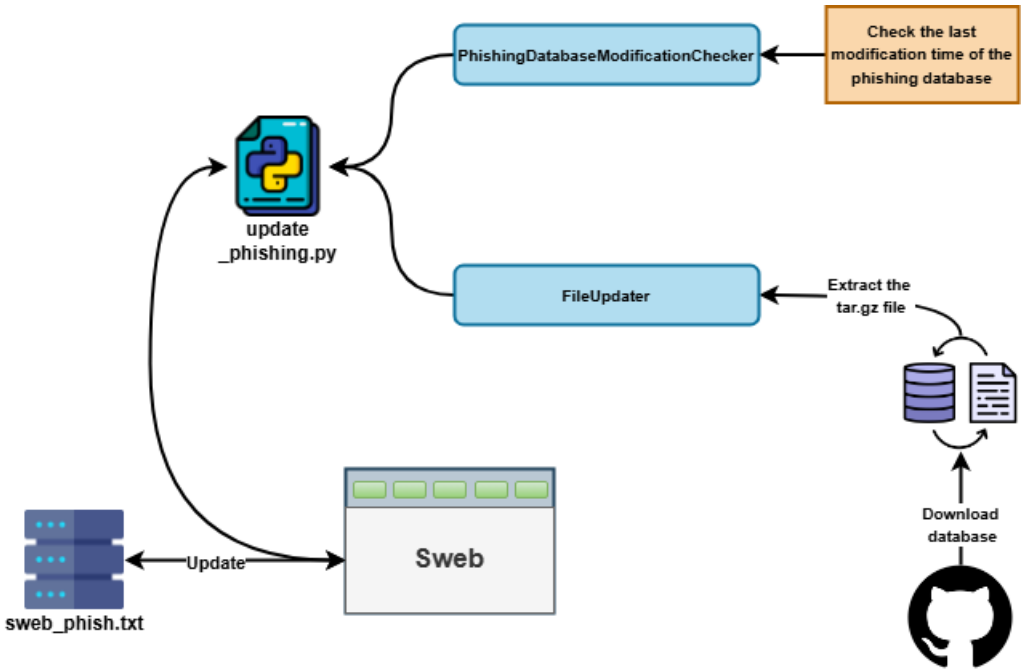


Fig. 6.2: Architecture of the fraudulent URL update mechanism.

Monitoring and Managing the Database Update Cycle

The `PhishingDatabaseModificationChecker` class is responsible for monitoring changes to the local phishing database and initiating updates when necessary. It includes methods for checking the last modification time of the database file and downloading an updated version from a remote source—in this case, a GitHub repository hosting a `.tar.gz` archive of the phishing data.

Listing 6.4: Initializing the `PhishingDatabaseModificationChecker` class.

```
class PhishingDatabaseModificationChecker:
    def __init__(self,sweb_config_data):
        # Get path to file SWEB_PHISH_1.txt
        self.path_to_phishing_database = sweb_config_data.phishingDatabase
        url_to_tar_github = sweb_config_data.phishingGithubDatabase
        self.database_updater = FileUpdater(url_to_tar_github,
            self.path_to_phishing_database)
```

Upon initialization, as shown in Listing 6.4, the class constructor (`__init__`) accepts two parameters, the instance itself `self` and configuration data. The configuration object provides paths to both the local phishing database and the remote URL for the compressed archive. This configuration is defined within the `config.json` file . A new instance of the `FileUpdater` class is also initialized using

the provided remote URL. This class is tasked with downloading and extracting the phishing site database from the `.tar.gz` archive hosted on GitHub.

Checking for File Modifications

To determine whether an update is necessary, the `get_last_modification_time` method is used. This method retrieves the last modification timestamp of the local phishing database and returns it as a `datetime` object. It verifies the existence of the database file using Python's `os.path.exists()` method. If the file is missing, it raises a `FileNotFoundError`. The method is listing in 6.5.

Listing 6.5: Retrieving the last time modification of the fraudulent database.

```
def get_last_modification_time(self):
        # Returns as a datetime object.
        # Check if the file is existed
        if not os.path.exists(self.path_to_phishing_database):
            print("Phishing database not found, The application will not
                update the database")
            pass
        else:
            # Get the last modificated time
            last_update_time =
                os.path.getmtime(self.path_to_phishing_database)
            # Using fromtimestamp to change it to Calender
            return datetime.fromtimestamp(last_update_time)
    # Returns True if modified after check_time, False otherwise.
    def file_has_been_modified_since(self, compared_time):
        return self.get_last_modification_time() > compared_time
```

The comparison of the last modification time is handled by the `file_has_been_modified_since` method. In this thesis, the threshold is set to two weeks. If the database has not been modified within that time frame, the method returns True, signaling that an update is required. Otherwise, it returns False.

Automated Update Execution

The `check_and_update_if_needed` method, shown in Listing 6.6, orchestrates the update process. It first calculates the two-week threshold using `datetime.now()` combined with a `timedelta` object. If the modification check returns True, the method triggers the `download_and_extract` function from the `FileUpdater` class to fetch and apply the latest phishing site database.

Listing 6.6: Method of accessing the database update.

```
def check_and_update_if_needed(self):
    if not os.path.exists(self.path_to_phishing_database):
        print("Phishing database not found, The application will not
            update the database")
        pass
    else:
        two_weeks_ago = datetime.now() - timedelta(weeks=2)
        if not self.file_has_been_modified_since(two_weeks_ago):
            self.database_updater.download_and_extract_from_github()
```

In summary, the `PhishingDatabaseModificationChecker` automates the process of evaluating and updating the phishing URL database. By ensuring the list is refreshed at least every two weeks, the Sweb browser maintains an up-to-date line of defense against known fraudulent sites an essential component in protecting users, especially seniors, from phishing attacks.

Downloading and Updating the Fraudulent Site Database

The `FileUpdater` class is responsible for downloading and updating the fraudulent site database from a remote source, specifically from a GitHub repository. It manages downloading a `.tar.gz` file, extracting its contents, and replacing the existing local database with the updated version.

The class is initialized via its `__init__` constructor, as shown in Listing 6.7. The constructor takes three input parameters: `self`, the URL `path to the remote .tar.gz` file on GitHub, and the local path to the fraudulent site database. These paths are retrieved from the `PhishingDatabaseModificationChecker` class and are defined in the `config.json`. The overall design of this class is illustrated in Figure 6.3.

Listing 6.7: FileUpdater class for downloading a database.

```
class FileUpdater:
    def __init__(self, github_url, path_to_database):
        #self.url_logger = input_url_logger
        self.github_url = github_url
        self.txt_path = path_to_database
        self.max_attempts = 2
        # Set delay after redirect HTTP
        self.delay_betwween_attempts = 0.1
```

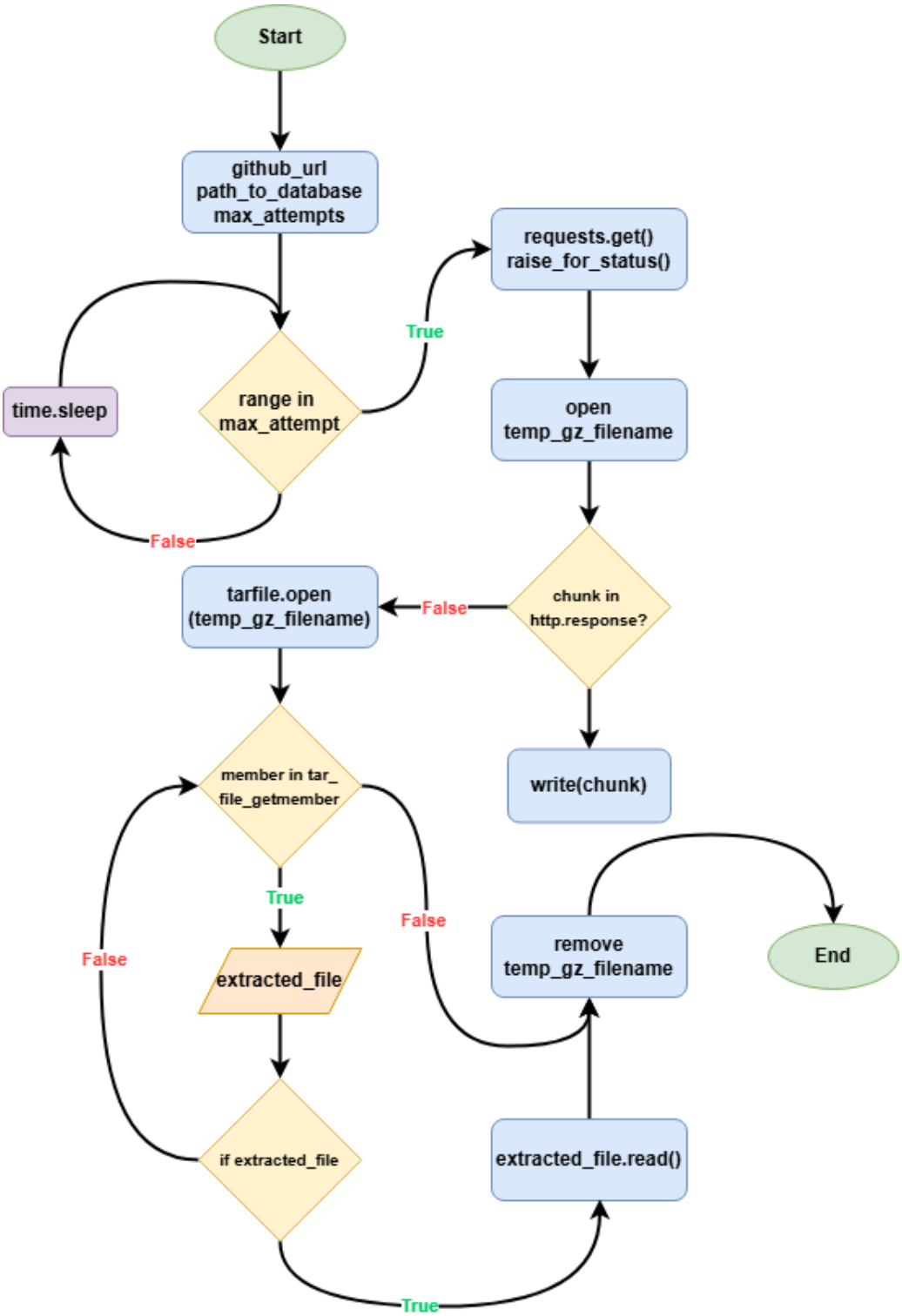


Fig. 6.3: Diagram for a class to store and update the database of a fraudulent site.

To ensure downloading, two parameters are defined: `max_attempts` and `delay_between_attempts`. These values determine how many times the application will retry the download in case of failure, and how long it waits between attempts,

respectively. This mechanism helps improve the reliability of the update process by handling intermittent network issues.

The core functionality is implemented in the `download_and_extract` method (see the first part of Listing 6.8). This method sends an `HTTP GET` request to the provided GitHub URL using the `requests.get` function. It passes `stream=True` to download the file in chunks, which is particularly useful for handling large files efficiently.

After the request is sent, the method verifies that the response was successful by calling `raise_for_status()`, which ensures that only responses with a 200 status code proceed further. The contents of the response are written to a temporary file using the `"wb"` (write-binary) mode. The file is written in chunks of 1024 bytes using the `iter_content` method, allowing efficient and memory-safe storage of the downloaded data.

Listing 6.8: Download and extraction method.

```
def download_and_extract_from_github(self):
    for attempt in range(self.max_attempts):
        try:
            # Connect to file_github and download
            http_response = requests.get(self.github_url, stream=True)
            # HTTPError object if an error has occurred during the process.
            http_response.raise_for_status()
            # Set file temp
            temp_gz_filename = "downloaded_phishing_database_temp.tar.gz"
            # Write to file temp
            with open(temp_gz_filename, "wb") as temp_file:
                for chunk in http_response.iter_content(chunk_size=1024):
                    temp_file.write(chunk)
            # Extract file to .txt
            try:
                with tarfile.open(temp_gz_filename, "r") as tar_file:
                    for member in tar_file.getmembers():
                        extracted_file = tar_file.extractfile(member)
                        if extracted_file:
                            with open(self.txt_path, "wb") as txt_file:
                                txt_file.write(extracted_file.read())
                # Remove file temp
                os.remove(temp_gz_filename)
                # Break for if the first HTTP is succeed
                break
            except tarfile.ReadError:
```

```
27                 pass
28         except (requests.ConnectionError, requests.HTTPError) as excep:
29             if attempt < self.max_attempts -1:
30                 # Wait for the specified delay before retrying
31                 time.sleep(self.delay_betwween_attempts)
32             else:
33                 pass
```

Once the temporary `.tar.gz` file is fully written, it is opened using Python's `tarfile` module. The contents are extracted using the extractfile method, which returns the file object if it exists. If the file is valid, it is written to the actual fraudulent site database path in binary mode `("wb")`. After a successful update, the temporary file is deleted using `os.remove()` to free up disk space and avoid unnecessary file clutter. The complete extraction logic is shown in the last part of Listing 6.8.

In summary, the `FileUpdater` class provides an efficient mechanism to keep the fraudulent site database current by downloading the latest version from a remote server and replacing the old one. This helps ensure the web browser has access to the most recent threat data, improving protection against phishing websites and enhancing overall user security.

6.3 Implementing Machine Learning for Phishing Detection

To enhance the security of the operating system designed for seniors, this chapter explores the implementation of machine learning techniques for phishing URL detection, aligning with current trends in cybersecurity, where phishing attacks through email and web communication are increasingly common.

In this project, a machine learning model was trained separately based on application runtime data and later integrated into the Sweb application. The trained model operates in a `"retrospective"` manner, it analyzes logs generated during web browsing sessions, identifies potentially dangerous URLs, and predicts whether these addresses represent genuine threats. This predictive layer provides an additional line of defense, improving overall browser safety.

This section is organized into the following topics:

Section 6.3.1: How the machine learning model was developed and trained before deployment.

Section 6.3.2: How the trained model was embedded into the Sweb application.

Section 6.3.3: How the model checks URLs and classifies them as safe or phishing threats.

6.3.1 Preparing the Trained Model

Due to the significant computational requirements of machine learning model training, the phishing detection model was trained separately, outside of the sweb application environment. Training machine learning models can demand substantial hardware resources, which may not be available on lightweight systems, such as the live ISO environment where the sweb application is intended to run. Therefore, based on the recommendation of the thesis advisor, the model was trained externally and only the finalized, trained model was later integrated into the sweb application.

The model was trained using a large dataset containing over `400,000` URLs, each labeled as either phishing `(True)` or legitimate `(False)`. The dataset format followed a simple structure, as illustrated below 6.1:

Table 6.1: Machine learning training dataset

url	**phish**
`https://www.smsticket.cz/mista/1379-kaple-boziho-tela`	False
`https://festivaly.eu/en/mezi-bloky-2011`	False
`https://netbank.trifinect.org/info.html`	True
`https://www.vhodne-uverejneni.cz/`	False
`http://whatsapweb.cyou`	True
`https://uch83.pages.dev`	True
`https://hotelopera.cz/sluzby/`	False
...	...

Each URL was paired with a corresponding label indicating whether it is considered fraudulent. The following listing 6.9 shows the Python code used for training the model on the `alldataset` dataset.

Listing 6.9: Python code for training machine learning.

```
import pandas as pd
import joblib
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix
```

```
7
8  # Step 1: Load the dataset (Ensure 'phishing_url_dataset.csv' is in the
       same directory)
9  data = pd.read_csv('/home/student/ML/PhisingSiteURL/alldataset.csv')
10 print("Dataset loaded successfully.")
11
12 # Convert 'phish' column from True/False to 1/0
13 data['label'] = data['phish'].astype(int)
14
15 # Step 2: Split the data into training and testing sets
16 X = data['url'] # Features (URLs)
17 y = data['label'] # Labels (0 for legitimate, 1 for phishing)
18 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
       random_state=42, stratify=y)
19 print("Data split into training and testing sets.")
20
21 # Step 3: Convert URLs to TF-IDF features
22 tfidf_vectorizer = TfidfVectorizer(max_features=5000)
23 X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
24 X_test_tfidf = tfidf_vectorizer.transform(X_test)
25 print("URLs converted to TF-IDF vectors.")
26
27 # Step 4: Train a Logistic Regression model
28 model = LogisticRegression()
29 model.fit(X_train_tfidf, y_train)
30 print("Model training complete.")
31
32 # Step 5: Make predictions
33 y_pred = model.predict(X_test_tfidf)
34
35 # Step 6: Evaluate the model
36 accuracy = accuracy_score(y_test, y_pred)
37 print(f"Accuracy: {accuracy:.2f}")
38 print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
39 print("\nClassification Report:\n", classification_report(y_test, y_pred))
40
41 # Step 7: Save the model and vectorizer for future use
42 joblib.dump(model, 'phishing_detection_model.pkl')
43 joblib.dump(tfidf_vectorizer, 'tfidf_vectorizer.pkl')
44 print("Model and vectorizer saved.")
```

At the beginning, the necessary Python libraries are imported:

- `pandas` for data handling.
- `joblib` for model serialization.
- `scikit-learn` modules for data preprocessing, model training, and evaluation.

The following is an explanation of each step in the code

- Step 1: The dataset is loaded into a pandas `DataFrame`. Then, the `phish` column is converted from `True/False` values into numeric labels (1 for phishing, 0 for legitimate) to make it suitable for machine learning algorithms.
- Step 2: The dataset is split into a training set `(80%)` and a testing set `(20%)` while maintaining the original class distribution (stratified sampling).
- Step 3: Each URL is transformed into a numerical representation using `TF-IDF` (Term Frequency-Inverse Document Frequency), limited to the top 5,000 features, to capture meaningful patterns in the URL strings.
- Step 4: A Logistic Regression model is trained on the `TF-IDF-transformed` URLs, learning to distinguish between legitimate and phishing URLs.
- Step 5: The trained model predicts whether each URL in the test set is `phishing` or `legitimate`.
- Step 6: The model's performance is evaluated using three metrics:
  - `Accuracy`: Proportion of correctly classified URLs.
  - `Confusion Matrix`: Breakdown of true positives, true negatives, false positives, and false negatives.
  - `Classification Report`: Includes precision, recall, and F1-score for both classes.
- Step 7: After training, both the model and the `TF-IDF` vectorizer are saved to disk using `joblib`, so they can be loaded later by the sweb application without needing to retrain.

The output of the training script includes two saved files:

1. `phishing_detection_model.pkl` (the trained logistic regression model).
2. `tfidf_vectorizer.pkl` (the `TF-IDF` vectorizer for transforming URLs).

These files are later imported into the sweb application to enable real-time phishing detection without the need for additional training.

6.3.2 Integrating Machine Learning into Sweb

The trained machine learning model created in the previous step was transferred into the sweb application and stored in the `ml` directory. The storage structure, including the location of the model files, is illustrated in Figure 6.4.

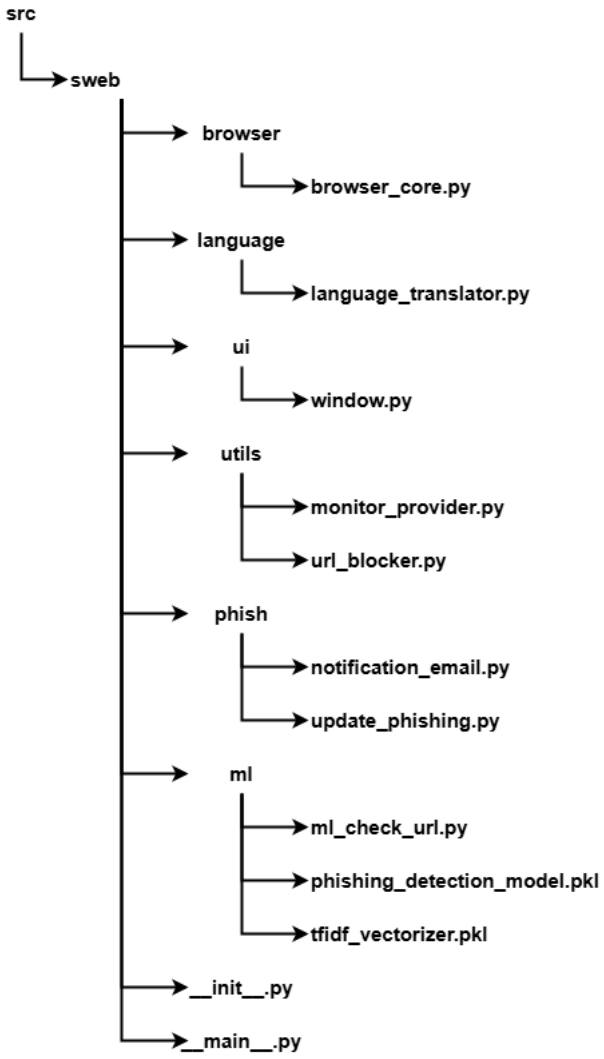


Fig. 6.4: SWEB application directory structure overview.

Inside the same `ml` directory, there is a Python script responsible for using the trained model to determine whether a given URL is likely to be a phishing site. This script is presented in Listing 6.10. It loads the pre-trained machine learning model and `TF-IDF` vectorizer and uses them to analyze and classify URLs.

Listing 6.10: Checking URL using machine learning.

```
import joblib
class PhishingURLDetector:
    def __init__(self, phishingDetectionModel, phishingVectorizer):
        # Load the trained model and the TF-IDF vectorizer
        try:
            self.model = joblib.load(phishingDetectionModel)
            self.vectorizer = joblib.load(phishingVectorizer)
        except:
```

```
            print("Error loading model and vectorizer")


    def is_phishing_url(self, url):
        url_tfidf = self.vectorizer.transform([url]) # Convert URL to
            TF-IDF features
        prediction = self.model.predict(url_tfidf) # Predict using the
            trained model
        return prediction[0] == 1 # Return True if the prediction is
            phishing, False otherwise
```

The script defines a class that offers a simple and efficient method for phishing URL detection. The main functionality revolves around verifying if a URL is suspicious (phishing) or legitimate, returning a Boolean result (True for phishing and False for legitimate).

Below is a detailed description of the code and its output:

1. Initialization (`__init__`): Upon creating an instance of `PhishingURLDetector`, the class attempts to load two important files:
   - The trained phishing detection model (a logistic regression model).
   - The `TF-IDF` vectorizer (used to convert URLs into numerical features).

   These files are loaded using the `joblib` library. If an error occurs during loading (e.g., missing file, wrong path), an error message is printed.
2. `is_phishing_url` Method:
   - Takes a single URL as input.
   - First, the URL is transformed into `TF-IDF` features using the loaded vectorizer.
   - The features are then passed to the trained model, which predicts whether the URL is phishing or not.
   - The method returns True if the model predicts the URL as phishing (1), and False otherwise.

The `is_phishing_url` function is called only after a URL has passed the blacklist detection phase. If a URL is already found on a predefined blacklist, it is immediately flagged as phishing without needing to invoke the machine learning model, it is showen in figure5.9. If a URL is not present in the blacklist, only then is the machine learning model used as a second line of defense to predict if the URL is suspicious. This approach optimizes performance by avoiding unnecessary model predictions and ensures quick detection for known threats.

The output of using the `PhishingURLDetector` class is straightforward:

- `True`: if the URL is detected as phishing.
- `False`: if the URL is considered legitimate.

Thus, the model provides a clean Boolean output that integrates easily into the rest of the security logic within the sweb application.

6.3.3 Verifying URL Threats

The `is_phishing_url` method from the `PhishingURLDetector` class is directly integrated into the browser's URL security verification mechanism. Specifically, it is utilized inside the `security_against_phishing` method (listing in 6.11), which is responsible for detecting phishing threats based on both blacklist detection and machine learning analysis.

Listing 6.11: Phishing URL detection based on blacklisting and machine learning.

```
def security_against_phishing(self,qurl):
    # Get url from QURL
    url_in_browser_value = qurl.toString()
    if url_in_browser_value.endswith('/'):
        if self.url_blocker.is_url_blocked(url_in_browser_value):
            self.toggle_phishing_webpage = True
            self.main_browser.setUrl(QUrl(url_in_browser_value))
            self.listen_for_keypress = True
            self.key_pressed_after_flag = False
            self.key_press_counter = 0

        else:
            if url_in_browser_value.startswith("https://www.") or
                url_in_browser_value.startswith("http://www."):
                if self.ml_phishing_url_detector.is_phishing_url
                (url_in_browser_value):
                    self.toggle_phishing_webpage = True
                    self.main_browser.setUrl(QUrl(url_in_browser_value))
                    self.listen_for_keypress = True
                    self.key_pressed_after_flag = False
                    self.key_press_counter = 0
        ...
        ...

        elif self.url_blocker.is_url_blocked(url_in_browser_value):
            self.toggle_phishing_webpage = True
            print("The third BL block condition is met")
            self.main_browser.setUrl(QUrl(url_in_browser_value))

```

```
            self.listen_for_keypress = True
            self.key_pressed_after_flag = False
            self.key_press_counter = 0
        else:
            if url_in_browser_value.startswith("https://www.") or
                url_in_browser_value.startswith("http://www."):
                if self.ml_phishing_url_detector.is_phishing_url
                (url_in_browser_value):
                    self.toggle_phishing_webpage = True
                    print("The fourth ML block condition is met")
                    self.listen_for_keypress = True
                    self.key_pressed_after_flag = False
                    self.key_press_counter = 0

    self.main_browser.loadFinished.connect(self.finished_load_web_page)
```

When a user attempts to access a URL, the `security_against_phishing` function is called, receiving the URL (encapsulated as a `QUrl` object) as input. The function first extracts the string representation of the URL from the `QUrl` object.
The verification process proceeds in two main phases:

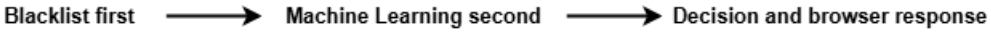
1. Blacklist Check: If the URL matches an entry in the internal blacklist (handled by `self.url_blocker.is_url_blocked()`), it is immediately flagged as phishing without further evaluation. In this case, the browser triggers appropriate defensive mechanisms, such as resetting keypress counters, blocking navigation, and updating the user interface to warn the user.
2. Machine Learning Model Check: If the URL is not found in the blacklist but still meets certain structural conditions (such as starting with `"http://www."` or `"https://www."`), the system applies a second layer of defense by calling the `self.ml_phishing_url_detector.is_phishing_url(url_in_browser_value)` method. This method uses the pre-trained machine learning model to predict whether the URL is likely phishing or legitimate based on the learned patterns from the training phase. If the model predicts the URL as phishing, the browser responds similarly to the blacklist detection: navigation is blocked, the phishing alert UI is triggered, and keypress monitoring is activated.

If neither the blacklist nor the machine learning model identifies the URL as phishing, the browser allows the page to load normally and resets its UI to the default safe style.

Throughout this process, the machine learning detection `is_phishing_url` is only engaged after the blacklist check has been passed. This ensures efficient use of resources by avoiding unnecessary `ML` evaluations for URLs already known to be

phishing through traditional methods.

The main logic flow can be summarized as:



Thus, the `security_against_phishing` function provides a layered security approach, combining traditional URL blacklists with modern machine learning techniques to enhance the protection of users from evolving phishing threats.

7 Advanced Web Browser Security for Seniors

The previous section introduced the fundamental methods for protecting users against fraudulent websites. Building upon that foundation, this chapter presents advanced security mechanisms aimed at safeguarding users' personal information. If a user attempts to enter information on a website identified as fraudulent within the database, Sweb will alert email which containing the website's link to the guardian. This additional security mechanism is illustrated in Figure 7.2.

Section 7.1 introduces the protection levels that sweb provides to secure users during browsing.

Section 7.2 focuses on the process of transmitting alerts and recorded user information from fraudulent sites to the custodian.

7.1 Protection Levels Provided by Sweb

Sweb offers three distinct protection levels: Level 1, Level 2, and Level 3. The protection level setting is obtained through the `global_configuration_provider`, where its value can be adjusted using the `sconf` application, a configuration tool that is part of the Senior Operating System project. The process of retrieving the protection level value is demonstrated in Listing 7.1.

Listing 7.1: Providing protection level by `global_dataProvider`.

```
self.global_dataProvider = global_dataProvider
...
...
if self.global_dataProvider.protectionLevel != 3:
    self.main_browser.setUrl(QUrl("about:blank"))
    self.url_toolbar.setVisible(not self.url_toolbar.isVisible())
    self.toolbar_space.setVisible(not self.toolbar_space.isVisible())
    self.menu_1_toolbar.setStyleSheet(self.default_style_toolbar())
    self.menu_2_toolbar.setStyleSheet(self.default_style_toolbar())
...
...
if self.global_dataProvider.protectionLevel == 2:
           web_url =
               self.url_blocker.find_url_with_value(url_in_bar_value)
```

Each protection level is described below:

Level 1 – Full Internet Access with Monitoring

At Protection Level 1, the user is allowed to use all predefined buttons located on the toolbar. Additionally, the search button is available, enabling the user to input any URL manually. Every manually entered link is verified through both blacklist detection and machine learning (ML) phishing detection. However, links associated with the predefined toolbar buttons are not subjected to blacklist or ML checks. This is because these links are pre-approved by the custodian (guardian), ensuring they are safe, and thus saving time by bypassing redundant checks.

In summary, at Level 1, users can browse the internet freely, either by clicking predefined buttons or by manually searching any web page.

Level 2 – Restricted Search to Trusted Websites

At Protection Level 2, users can still use all the predefined toolbar buttons without limitation. The search functionality remains available but is restricted to trusted websites only, meaning the user can search for and access only sites that are pre-approved and added to a whitelist using the sconf application.

If a user attempts to access a site not listed in the whitelist, Sweb displays a message indicating that the page is not allowed. It is sufficient for the user to enter a partial name or URL, Sweb will then attempt to open the first matching or partially matching trusted website. The implementation of this search validation is detailed in Listing 7.2.

Listing 7.2: Search validation mechanism used in Sweb for protection level 2.

```
def load_urls_from_allowed_website(self, path):
    try:
        with open(path,"r") as openfile:
            # Real all member from the file until the file is None
            if openfile is not None:
                content = openfile.read()
                reading_url = content.strip().split('\n')
                self.permitted_website_list.update(reading_url)
    except FileNotFoundError:
        print("Permitted website list not found, The application will
            continue to run without blocking any URLs")
        pass
def find_url_with_value(self, search_value):

    for line in self.permitted_website_list:
        line = line.strip()
```

```
16          if search_value in line:
17              return line
18      return False
```

The above code shows the implementation of the search validation mechanism used in Sweb for protection level 2. It contains three main functions:

- `load_urls_from_allowed_website(self, path)`: Similarly, this function loads a list of trusted or permitted websites into the application's internal database `permitted_website_list`. These trusted websites are allowed during browsing at protection level 2. If the permitted list file is not found, a warning is printed, and the application continues running normally.
- `find_url_with_value(self, search_value)`: This function enables flexible searching by checking whether a user-entered value (either part of a URL or a name) matches any trusted website in the permitted list. It goes through each stored URL, and if the `search_value` is found anywhere in a trusted URL, it returns the matching URL. If no match is found, it returns False. This allows the user to enter only a partial URL or keyword instead of the full address.

This code ensures that Sweb can validate user input against a list of trusted websites quickly and flexibly, improving security while maintaining ease of use at protection level 2.

Level 3 – Strict Access to Predefined Pages Only

Protection Level 3 is the most restrictive setting. At this level, the search functionality is disabled entirely, and users can only access web pages linked directly through the predefined toolbar buttons. However, if a user clicks any additional links on those allowed pages, the new URLs will still be checked against the blacklist and processed through ML phishing detection before loading.

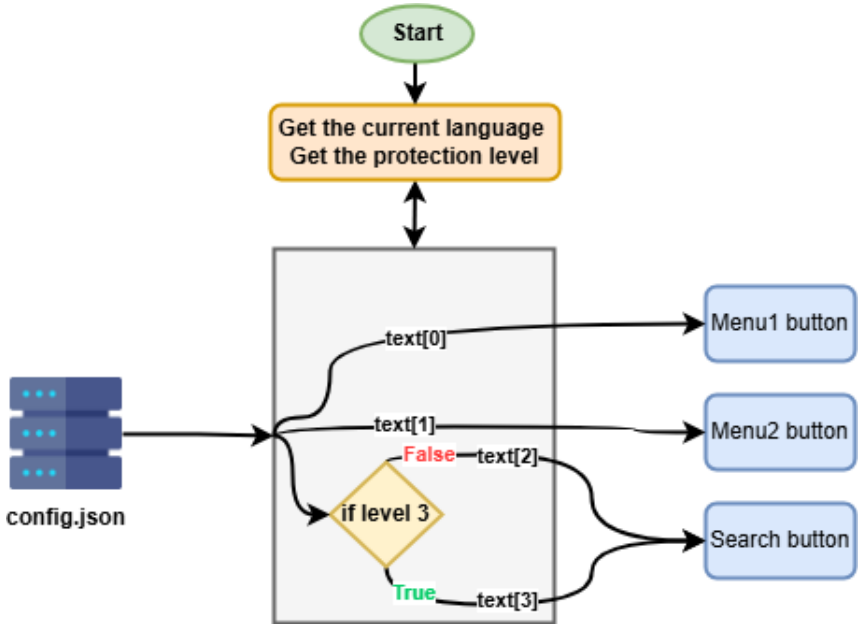


Fig. 7.1: Setting the search button based on the protection level.

Listing 7.3: Description of code for setting search button label based on protection level.

```
def get_translated_text(self, button_name):
    key_text = f"sweb_{self.current_language_in_browser}_{button_name}"
    #print("text",self.text[1])
    #return self._dataProvider.text

    if key_text == "sweb_en_menu1":
        return self.text[0]
    if key_text == "sweb_en_menu2":
        return self.text[1]
    if key_text == "sweb_en_menu2Address":
        if self.protection_level == 3:
            return self.text[3]
        return self.text[2]
```

The code 7.3 provided is a method that determines the text label to display for a search button, depending on the current protection level of the Sweb browser. The figure 7.1 shows the method of setting the search button based on the protection level.

- `get_translated_text(self, button_name)`: This method takes a `button_name` as input and returns the appropriate label text based on the current protection level and language settings. It dynamically selects the right

label for the button, making the text responsive to both the language and the security configuration.

- Key Variables and Logic: The `key_text` is generated by concatenating the prefix `"sweb_"`, the current language in the browser `(self.current_language_in_browser)`, and the name of the button `button_name`. This constructs a unique key for each button's label depending on the language setting and button type. The code checks if the `key_text` matches specific predefined values (for example, `sweb_en_menu2Address`) for the search button's label.
    1. If the protection level is 3 (the strictest level), the label for the search button is set to the value in `self.text[3]`. This likely means that the search button is either disabled or carries a different label, reflecting restricted access.
    2. For protection levels below 3, the label is set to `self.text[2]`, which is the default behavior.

This method ensures that the search button's text adapts based on the current protection level. When level 3 is active, the text label changes (and the button might be disabled), signaling to the user that they cannot perform any searches and are only allowed to visit predefined, safe sites.

The code 7.4 provided code defines a method `toggle_url_toolbar(self)` that controls the visibility of the URL toolbar and toolbar space in the main browser, depending on the current protection level set in the system.

Listing 7.4: Disable opening input field.

```
def toggle_url_toolbar(self):
    """
    Toggles the visibility of the URL toolbar and the toolbar space in
        the main browser.
    """
    if self.global_dataProvider.protectionLevel != 3:
        self.main_browser.setUrl(QUrl("about:blank"))
        self.url_toolbar.setVisible(not self.url_toolbar.isVisible())
        self.toolbar_space.setVisible(not self.toolbar_space.isVisible())
        self.menu_1_toolbar.setStyleSheet(self.default_style_toolbar())
        self.menu_2_toolbar.setStyleSheet(self.default_style_toolbar())
```

- `toggle_url_toolbar(self)`: This method is responsible for toggling (showing or hiding) the URL toolbar and toolbar space in the browser. It does so by checking the current protection level and adjusting the visibility accordingly.
- `if self.global_dataProvider.protectionLevel != 3:` The visibility of the URL toolbar and the toolbar space is only toggled if the protection level

is not 3. If the protection level is set to 3, no changes are made, meaning the URL input field remains hidden, as users are restricted to navigating through predefined links only.

- `self.main_browser.setUrl(QUrl("about:blank"))` This line sets the browser's URL to "about:blank" when the toolbar is toggled, effectively clearing the URL field and preventing any new URL input.
- `self.url_toolbar.setVisible(not self.url_toolbar.isVisible())` This line toggles the visibility of the URL toolbar. If it's visible, it will be hidden; if it's hidden, it will be shown.
- `self.toolbar_space.setVisible(not self.toolbar_space.isVisible())` Similarly, this line toggles the visibility of the toolbar space, adjusting the layout accordingly.
- `Styling Updates:` The style for `menu_1_toolbar` and `menu_2_toolbar` is reset to the default, ensuring that the UI remains consistent after the visibility change.

The method ensures that when the protection level is set to 3, the URL input field (and associated toolbar) remains hidden, preventing the user from entering any new URLs or performing searches. This is a security measure to enforce that users can only interact with predefined, trusted websites.
Summary, When the protection level is set to 3, the search button is disabled, and its label is updated to reflect this restricted access, ensuring the user can only navigate through predefined links. Additionally, the `toggle_url_toolbar` method hides the URL input field and toolbar space, further restricting the user's ability to manually input URLs. These measures make the browser more secure by preventing users from navigating to unauthorized websites and limiting their interaction to only predefined, trusted links.

The following table 7.1 summarizes the key differences between the three protection levels offered by Sweb, highlighting how each level progressively restricts browsing flexibility to enhance user safety.

Table 7.1: The key differences between the three protection levels.

Feature	Level 1	Level 2	Level 3
Toolbar Buttons Access	Allowed	Allowed	Allowed
Search Field Availability	Enabled (any website)	Enabled (trusted websites only)	Disabled
Manual URL Entry	Any URL	Only whitelisted URLs	Not allowed
Blacklist and ML URL Checking	For all links	For all links	For all links
Trusted Sites Whitelisting	Not required	Required	Required for toolbar links
Flexibility	High	Medium	Very limited

7.2 Secure Transmission of Alerts and User Information to the Custodian

In Sweb, an important security feature is the ability to notify the custodian (guardian) when a senior user visits a phishing website and enters any sensitive information. This secure transmission of alerts is crucial because early warning allows the guardian to take timely action, such as contacting the senior user, changing compromised credentials, or providing further assistance to prevent potential financial or personal harm. Immediate alerts strengthen the overall protection of seniors by reducing the window of vulnerability after a phishing incident. The email address of the guardian is provided by the `global_configuration_provider`, where it is set in the `config.json` configuration file. The guardian's email can be configured and updated using the `sconf` application, which is a part of the Senior OS project. The code shown in Listing 7.5 demonstrates how the email address is assigned to the `NotificationFillTextToPhishing` class:

Listing 7.5: Assigning the guardian's email for phishing alert notifications.

```
1 self.notification_fill_text =
      NotificationFillTextToPhishing(self.sweb_dataProvider.
2     command_line_mail_script, self.global_dataProvider.careGiverEmail)
```

Rather than integrating a mailing service directly into Sweb, the application relies on the `smail` utility, a dedicated mailing tool included in the Senior OS project. Sweb invokes a prepared script from Smail to send the alert email. The path to the mail script is provided by `sweb_dataProvider.command_line_mail_script`, ensuring that the transmission process remains simple, modular, and reliable.

Once the guardian's email is set, Sweb must be able to send a warning message automatically when necessary. When sensitive information is entered on a phishing site, Sweb uses the following method to send an alert email in the background without interrupting the senior's browsing session. The implementation is shown in Listing 7.6.

Listing 7.6: Sending phishing alert email to the guardian.

```
def send_email(self, message_to_receiver):
    # Load needed configuration from sweb_config in sconf for sending
        notification to authorized people
    message = f"Subject: Phishing Warning, Senior visited the
        page:\n\n{message_to_receiver} \n\nPlease check the website and
        take necessary action."
    try:
        # Load the command line mail script
        # Send email using the command line mail script, it runs in the
            background
        subprocess.Popen(["python3", self.command_line_mail_script,
            self.careGiverEmail, message], stdout=subprocess.DEVNULL,
            stderr=subprocess.DEVNULL)

    except Exception as excep:
        print(f"Error sending email: {excep}")
```

The `send_email` method constructs an alert message containing information about the suspicious page visited by the user. It then launches the external email-sending script (smail) using Python's `subprocess.Popen`, providing it with the caregiver's email address and the message content as arguments. The use of `subprocess.Popen` allows the email to be sent asynchronously, meaning the main Sweb application remains responsive. Any errors encountered during the sending process are caught and printed for debugging purposes.

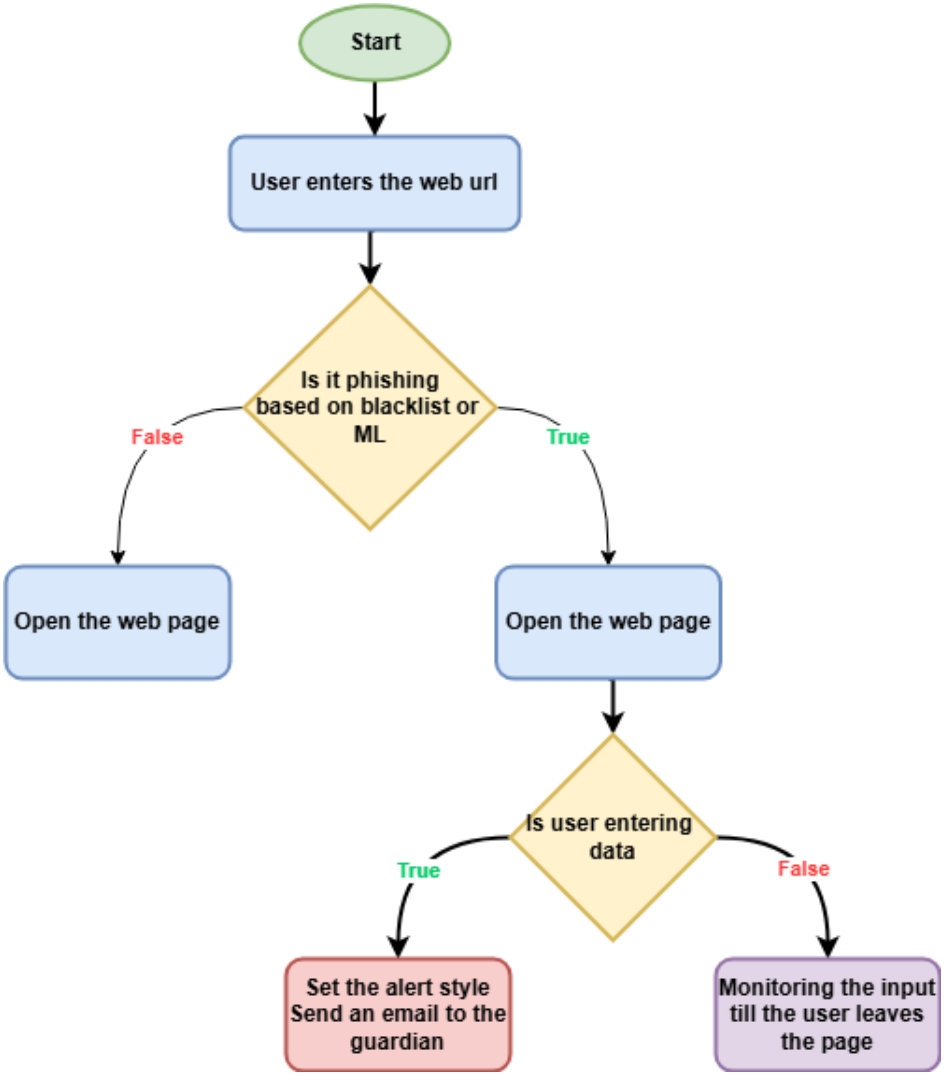


Fig. 7.2: Flow of email notification trigger after user input on a phishing website.

Once Sweb detects that a user has entered a phishing website, it does not immediately trigger a warning email. Instead, it waits for confirmation that the user is actively interacting with the page. This is done to avoid false alarms when a senior accidentally visits a phishing site without entering any personal data. The method is shown in Figure 7.2.

Listing 7.7: Key press event for phishing UI activation.

```
def keyPressEvent(self, event):
    if self.listen_for_keypress:
        self.key_press_counter += 1
        print(f"[{self.key_press_counter}] Key pressed: {event.key()} ({
            event.text()})")
        if self.key_press_counter >= 2:
            self.key_pressed_after_flag = True
```

```
            # Apply phishing UI changes here
            self.menu_1_toolbar.setStyleSheet(self.phishing_style_toolbar())
            self.menu_2_toolbar.setStyleSheet(self.phishing_style_toolbar())
            print("Phishing UI changes applied.")
            self.notification_fill_text.send_email(self.main_browser.url()
            .toString())
            # Optional: Stop listening after it triggers
            self.listen_for_keypress = False
```

In the code 7.7 the `keyPressEvent` method is responsible for monitoring the user's keystrokes once phishing behavior is suspected. If `listen_for_keypress` is enabled, each key press increments the `key_press_counter`. When the counter reaches 2 or more, meaning the user has typed at least two characters, the system assumes that sensitive information may be getting entered.

At this point:

- A visual alert is triggered by changing the toolbar's style to a phishing warning style.
- An email notification is sent to the guardian using the `send_email` method, including the current page's URL.
- Further `keypress` monitoring is disabled to prevent multiple notifications for the same event.

This mechanism ensures that an alert is sent only when there is a real risk, balancing security with usability.

8 Challenges Encountered During Thesis Development

Throughout the development of this project, a number of technical and organizational challenges were encountered. This chapter outlines the main difficulties faced during the project and describes the solutions that were implemented to overcome them. Each challenge required careful consideration and often led to significant improvements in the design and functionality of the final solution.
The main challenges included:

Transitioning Configuration Management to a Central Data Provider

Initially, configuration parameters were loaded directly by reading from `JSON` files scattered throughout the application. To improve maintainability and scalability, a centralized `DataProvider` approach was introduced. This required extensive refactoring of the existing source code, ensuring that all parts of the application retrieved configuration data through a single, unified interface.

Advantages of using a centralized `DataProvider` include:

1. Easier maintenance and updates, as changes are made in one place.
2. Improved consistency across the application.
3. Better support for future extensions, such as dynamic configuration updates or loading from multiple sources.

Refactoring the Code Structure for Poetry Compatibility

Another major challenge was reorganizing the project structure to be fully compatible with the Poetry dependency management and packaging system. Adopting Poetry brought significant benefits, particularly in simplifying the integration of the Sweb application into the larger Senior-OS project.
Advantages of using Poetry include:

1. Streamlined dependency management and version control.
2. Simplified building and packaging of the application.
3. Easier deployment and integration into larger systems.
4. Better environment isolation, reducing dependency conflicts.

Training the Machine Learning Model for Phishing Detection

One of the most time-consuming challenges was obtaining a reliable and comprehensive dataset for training the phishing detection model. Finding datasets that clearly

labeled URLs as either phishing or legitimate was difficult, and significant effort was spent verifying, cleaning, and organizing the data to ensure its quality. Furthermore, balancing the dataset to avoid bias and fine-tuning the model for realistic performance added additional layers of complexity.
Specific challenges included:

1. Locating up-to-date, publicly available datasets.
2. Preprocessing the data to remove duplicates and inconsistencies.
3. Ensuring the dataset remained representative of real-world phishing attacks.

Despite these difficulties, the final trained model achieved satisfactory performance and contributed significantly to the effectiveness of the application.

9 Final Deliverables and User Documentation

As part of this diploma thesis, an electronic appendix has been created and is uploaded to the BUT information system in an appropriate format (e.g., ZIP, PDF). The appendix includes all necessary documentation and source files related to the Senior Web Browser project.

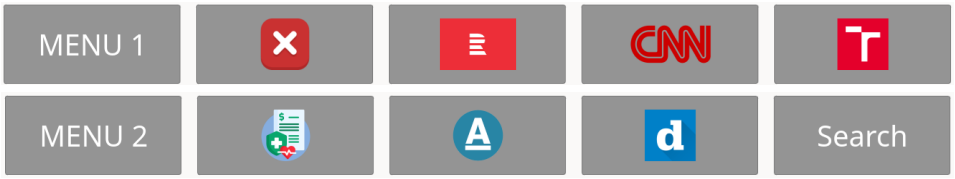
In addition to the source files and documentation, user manuals for the web browser have been created in English 9.1, Czech 9.3, and German 9.5 to ensure accessibility for seniors from various linguistic backgrounds. The manuals are written in a clear, simplified style, focusing on ease of understanding to meet the specific needs of elderly users. They provide step-by-step guidance for using the browser's core features, such as navigating web pages and recognizing security warnings about fraudulent websites.

Furthermore, the final version of the source code is publicly available on GitHub, where it can be downloaded and tested. The Figure 9.8 presents an overview of the GitHub repository's structure and its main components, illustrating the organization of the project files.

Instructions for using the SWEB

Controlling the application:

The application has a main control panel that contains 5 buttons per menu (10 buttons in total):



1. **Menu** - allows you to switch between two different parts of the application (Menu 1 and Menu 2).
2. **Red button with white cross** - used to exit the application.
3. **Buttons with website icons** - quick access to a specific website, one click and the website opens on the screen.
4. **Button "Search"** - opens a blank field where the user can enter any web address.

Opening a Website

- **Protection Level 1: Basic Protection**

1. **Opening via Menu:** Click any button or icon in Menu 1/2 to directly open the website.
2. **Opening via Search:** Click the search button, enter your search text, and the input will be checked against a "blacklist" and phishing detection using a neural network.

- **Protection Level 2: Enhanced Protection**

1. **Opening via Menu:** Click any button or icon in Menu 1/2 to directly open the website.
2. **Opening via Search:** Click the search button, enter your search text, and the input will be verified against a whitelist. The system will only open URLs that match the

Fig. 9.1: Manual written in English, page 1.

input in the whitelist.

- **Protection Level 3: Maximum Protection**

1. **Opening via Menu:** Click any button or icon in Menu 1/2 to directly open the website.
2. **No Search:** The search button will be disabled, and search functionality is not available at this protection level.

Phishing Detection Alert

- **Phishing Detection:**
  - Entering URL or Visiting a Site: When the user enters a URL or visits a website, the system checks if the URL is listed in the blacklist or if it is detected through the neural network for phishing.
- **Alert Mechanism:**
  - If a potential phishing threat is detected, the button on the menu bar will change color to red to alert the user about the potential danger.
- **Ignoring the Alert:**
  - If the user ignores the red alert and proceeds to enter any data on the website, the app will automatically send an email notification to the guardian to inform them about the suspicious activity.

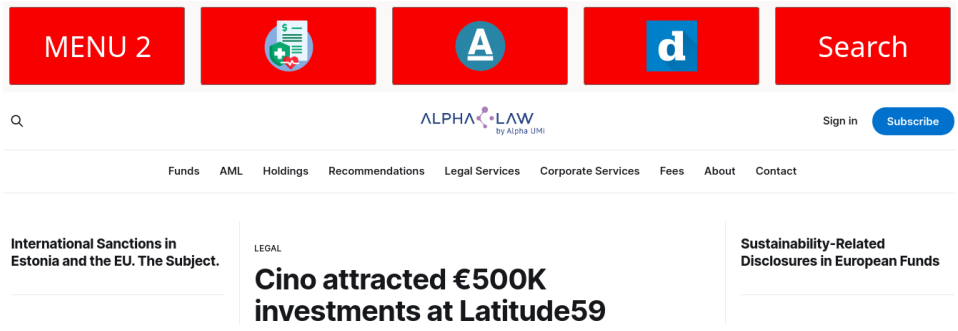
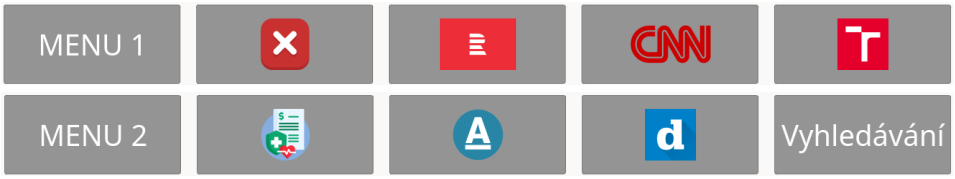


Fig. 9.2: Manual written in English, page 2.

Návod na použití aplikace SWEB

Ovládání aplikace:

Aplikace má hlavní ovládací panel, který obsahuje 5 tlačítek pro jedno menu (celkově 10 tlačítek):



1. **Menu** – umožňuje přepínat mezi dvěma různými částmi aplikace (Menu 1 a Menu 2).
2. **Červené tlačítko s bílým křížkem** – slouží k ukončení aplikace.
3. **Tlačítka s ikonami webů** – rychlý přístup na konkrétní webovou stránku, jedno kliknutí a webová stránka se otevře na obrazovce.
4. **Tlačítko "Vyhledávání"** – otevře prázdné pole, do kterého může uživatel zadat libovolnou webovou adresu.

Otevření webové stránky

- **Úroveň ochrany 1: Základní ochrana**

1. **Otevření prostřednictvím nabídky:** Kliknutím na libovolné tlačítko nebo ikonu v MENU 1/2 otevřete webovou stránku přímo.
2. **Otevření prostřednictvím vyhledávání:** Otevřete webovou stránku pomocí vyhledávání. Klikněte na tlačítko vyhledávání, zadejte hledaný text a zadání bude zkontrolováno podle blacklistu a detekce phishingu pomocí neuronové sítě.

- **Úroveň ochrany 2: Zvýšená ochrana**

1. **Otevření prostřednictvím nabídky:** Kliknutím na libovolné tlačítko nebo ikonu v MENU 1/2 se webová stránka otevře přímo.
2. **Otevření prostřednictvím vyhledávání:** Otevřete webovou stránku, kterou chcete

Fig. 9.3: Manual written in Czech, page 1.

otevřít: Klikněte na tlačítko vyhledávání, zadejte hledaný text a vstup bude ověřen podle bílé listiny. Systém otevře pouze ty adresy URL, které odpovídají zadání na Whitelistu.

- **Úroveň ochrany 3: Maximální ochrana**

1. **Otevření prostřednictvím nabídky:** Kliknutím na libovolné tlačítko nebo ikonu v MENU 1/2 se webová stránka otevře přímo.
2. **Žádné vyhledávání:** Tlačítko pro vyhledávání bude zakázáno a funkce vyhledávání není na této úrovni ochrany k dispozici.

Upozornění na detekci phishingu:

- **Detekce phishingu:**
  - Zadání adresy URL nebo návštěva webu: Při zadávání adresy URL nebo návštěvě webové stránky systém zkontroluje, zda je adresa URL uvedena na černé listině nebo zda je prostřednictvím neuronové sítě detekována jako phishingová.
- **Mechanismus upozornění:**
  - Pokud je detekována potenciální hrozba phishingu, tlačítko na panelu nabídky změní barvu na červenou, aby uživatele upozornilo na potenciální nebezpečí.
- **Ignorování výstrahy:**
  - Pokud uživatel ignoruje červené upozornění a pokračuje v zadávání jakýchkoli údajů na webové stránce, aplikace automaticky odešle e-mailové upozornění opatrovníkovi, aby ho informovala o podezřelé aktivitě.

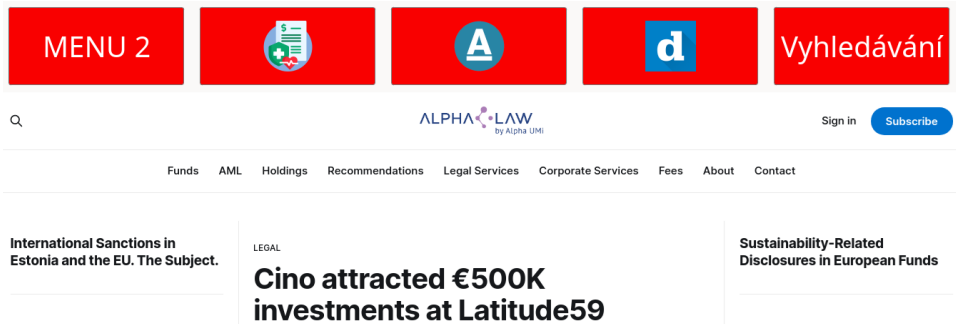
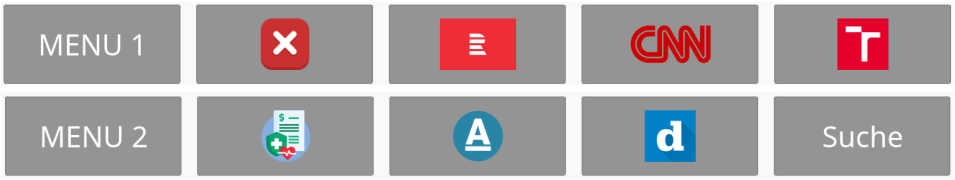


Fig. 9.4: Manual written in Czech, page 2.

Anleitung zur Nutzung der SWEB

Steuerung der Anwendung:

Die Anwendung verfügt über ein Hauptbedienfeld mit 5 Tasten pro Menü (insgesamt 10 Tasten):



1. **Menü** - ermöglicht es Ihnen, zwischen zwei verschiedenen Teilen der Anwendung zu wechseln (Menü 1 und Menü 2).
2. **Rote Taste mit weißem Kreuz** - dient zum Beenden der Anwendung.
3. **Schaltflächen mit Website-Symbolen** - schneller Zugang zu einer bestimmten Website, ein Klick und die Website öffnet sich auf dem Bildschirm.
4. **Schaltfläche "Suchen"** - öffnet ein leeres Feld, in das der Benutzer eine beliebige Webadresse eingeben kann.

Een website openen

- **Beschermingsniveau 1: Basisbescherming**

1. **Openen via menu:** Klik op een knop of pictogram in Menu 1/2 om de website direct te openen.
2. **Openen via Zoeken:** Klik op de zoekknop, voer uw zoektekst in en de invoer wordt gecontroleerd aan de hand van een “zwarte lijst” en phishingdetectie met behulp van een neuraal netwerk.

- **Beschermingsniveau 2: Verbeterde bescherming**

1. **Openen via menu:** Klik op een knop of pictogram in Menu 1/2 om de website direct te openen.

Fig. 9.5: Manual written in German, page 1.

2. **Openen via Zoeken:** Klik op de zoekknop, voer je zoektekst in en de invoer wordt geverifieerd aan de hand van een witte lijst. Het systeem opent alleen URL's die overeenkomen met de invoer in de witte lijst.

- **Beschermingsniveau 3: maximale bescherming**

1. **Openen via menu:** Klik op een knop of pictogram in Menu 1/2 om de website direct te openen.
2. **Geen zoekopdracht:** De zoekknop wordt uitgeschakeld en de zoekfunctie is niet beschikbaar op dit beschermingsniveau.

Waarschuwing voor phishingdetectie

- Phishing-detectie:
    - URL invoeren of website bezoeken: Wanneer de gebruiker een URL invoert of een website bezoekt, controleert het systeem of de URL is opgenomen in de zwarte lijst of via het neurale netwerk is gedetecteerd voor phishing.
- **Waarschuwingsmechanisme:**
    - Als een potentiële phishing-bedreiging wordt gedetecteerd, verandert de knop op de menubalk van kleur in rood om de gebruiker te waarschuwen voor het potentiële gevaar.
- **De waarschuwing negeren:**
    - Als de gebruiker de rode waarschuwing negeert en doorgaat met het invoeren van gegevens op de website, stuurt de app automatisch een e-mailmelding naar de bewaker om hen te informeren over de verdachte activiteit.

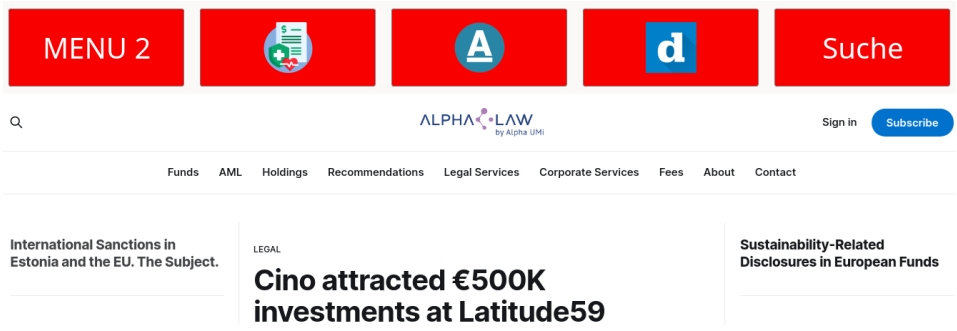
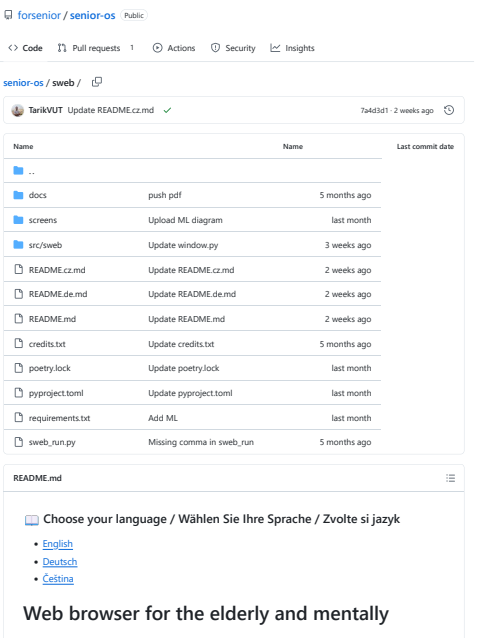
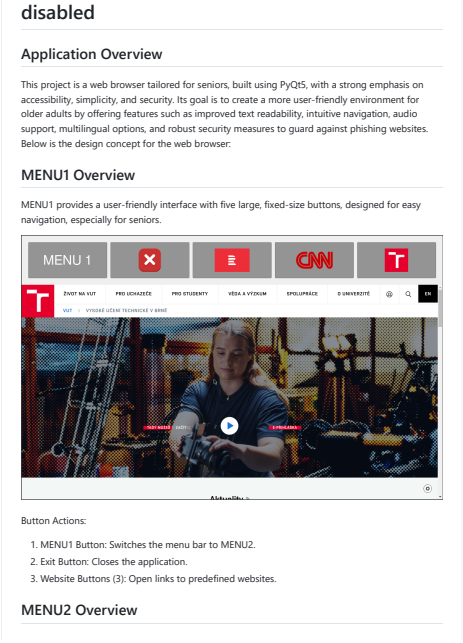


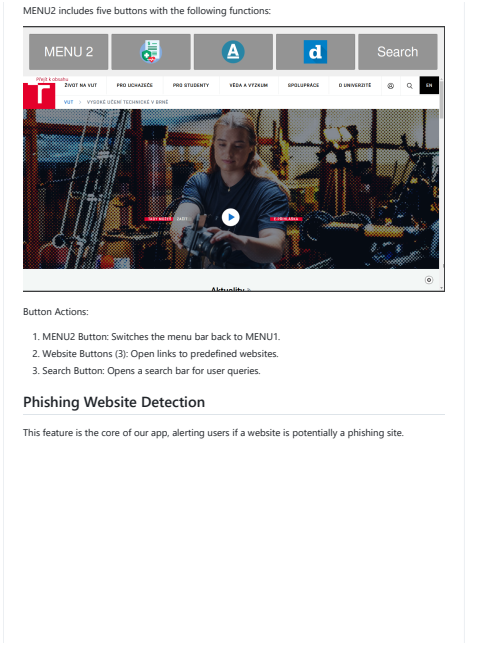
Fig. 9.6: Manual written in German, page 2.



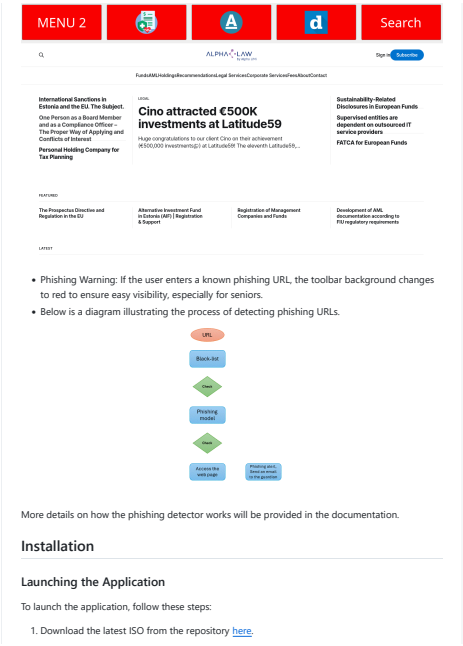
(a) Page 1.



(b) Page 2.

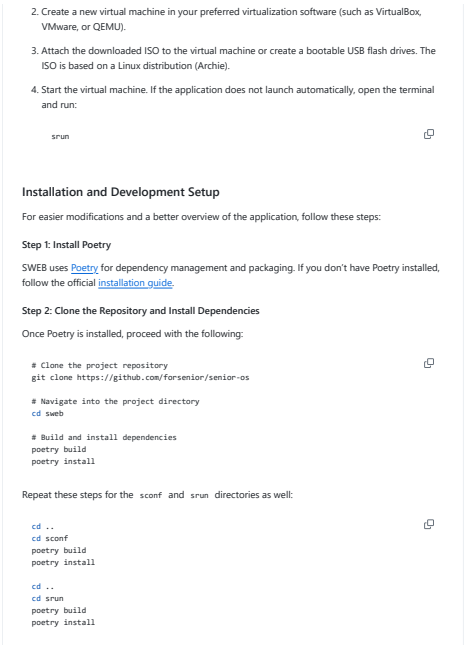


(c) Page 3.



(d) Page 4.

Fig. 9.7: GitHub repository overviews, pages 1 to 4.



2. Create a new virtual machine in your preferred virtualization software (such as VirtualBox, VMware, or QEMU).
3. Attach the downloaded ISO to the virtual machine or create a bootable USB flash drives. The ISO is based on a Linux distribution (Archie).
4. Start the virtual machine. If the application does not launch automatically, open the terminal and run:

```
srun
```

Installation and Development Setup

For easier modifications and a better overview of the application, follow these steps:

Step 1: Install Poetry

SWEB uses Poetry for dependency management and packaging. If you don't have Poetry installed, follow the official installation guide.

Step 2: Clone the Repository and Install Dependencies

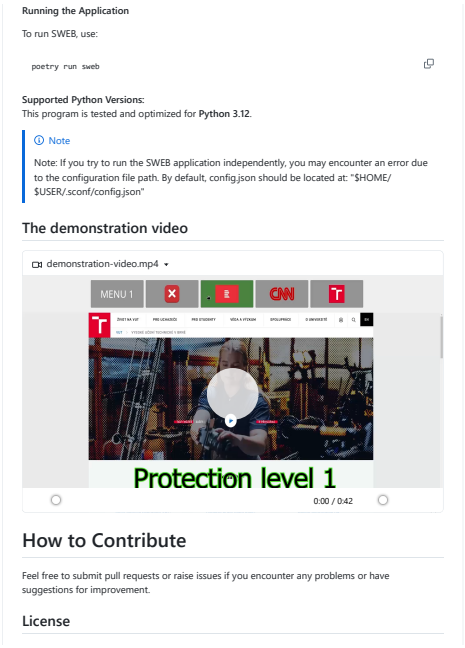
Once Poetry is installed, proceed with the following:

```
# Clone the project repository
git clone https://github.com/forsenior/senior-os

# Navigate into the project directory
cd sweb

# Build and install dependencies
poetry build
poetry install
```

Repeat these steps for the `sconf` and `srun` directories as well:

```
cd ..
cd sconf
poetry build
poetry install

cd ..
cd srun
poetry build
poetry install
```

(a) Page 5.



Running the Application

To run SWEB, use:

```
poetry run sweb
```

Supported Python Versions:
This program is tested and optimized for **Python 3.12**.

> Note
>
> Note: If you try to run the SWEB application independently, you may encounter an error due to the configuration file path. By default, config.json should be located at: "$HOME/$USER/.sconf/config.json"

The demonstration video

demonstration-video.mp4

How to Contribute

Feel free to submit pull requests or raise issues if you encounter any problems or have suggestions for improvement.

License

(b) Page 6.

Fig. 9.8: GitHub repository overviews, pages 5 to 6.

Conclusion

This diploma thesis, titled Web Browser for Seniors, focuses on the design, development, and enhancement of a user-friendly web browser specifically tailored for seniors, particularly those aged 90 and above. The main objective was to simplify everyday computer use while significantly improving accessibility and online safety for this age group.

The theoretical section of the thesis provides an in-depth analysis of fraudulent attacks, with a strong emphasis on phishing. It highlights the specific risks such attacks pose to senior users and discusses various detection techniques. Furthermore, it outlines the tools and technologies utilized in the development process, including graphics libraries, HTTP request handling libraries, system integration tools, and machine learning frameworks.

The practical section describes the development of the Senior Web Browser (Sweb) using Python. Key features implemented include:

- Multilingual support for better accessibility to non-English speakers.
- Adjustable text sizes and clear icons for enhanced readability and navigation.
- Phishing detection through a dynamic blacklist system.
- Integration of a machine learning model to improve phishing URL detection.
- Guardian notification system, where alerts are automatically sent if a senior user interacts with a phishing page.

Security enhancements were a major focus. The browser now verifies all clicked links against a blacklist and the machine learning model, and updates the blacklist database regularly to ensure protection against emerging threats. The notification system for guardians is securely managed via a trusted external script, improving reliability without embedding sensitive mail functions directly into the browser.

Moreover, the project modernized the code structure by introducing a centralized data provider for configuration management and adopting Poetry for dependency management and project organization. These changes improve maintainability and simplify integration with the broader Senior OS environment.

The machine learning model was trained and successfully integrated into Sweb, with extensive testing ensuring its reliability in live browsing scenarios.

Finally, the full source code and documentation are publicly available on GitHub, providing opportunities for further development, testing, and real-world deployment.

This work not only delivers a safer and more accessible browsing solution for seniors but also lays the foundation for future enhancements, including broader machine learning integration and expanding multilingual support.

Bibliography

[1] Fortinet, Inc. *Different Types of Phishing Attacks.* Available from: https://www.fortinet.com/resources/cyberglossary/types-of-phishing-attacks. [cite. 2024-11-27].

[2] Holly Dowden. *Social Engineering Attacks.* Available from: https://www.ntiva.com/blog/social-engineering-attacks-how-to-protect-yourself. [cite. 2024-11-29].

[3] Alabdan, Rana. *Phishing attacks survey: Types, vectors, and technical approaches.* [cite. 2024-11-23].

[4] Create GUI Applications with Python & Qt5 (PyQt5 Edition): *The hands-on guide to making apps with Python. Dr Martin Fitzpatrik* [cite. 2024-11-23].

[5] Riverbank Computing Limited *PyQt5 - Comprehensive Python Bindings for Qt v5* Available from: https://pypi.org/project/PyQt5/#:~:text=PyQt5%20is%20a%20comprehensive%20set,platforms%20including%20iOS%20and%20Android.. [cite. 2024-11-27].

[6] Martin Fitzpatrick *Custom Qt5 Python Widgets* Available from: https://pypi.org/project/qtwidgets/. [cite. 2024-11-24].

[7] *PyQt5 – QApplication* Available from: https://www.geeksforgeeks.org/pyqt5-qapplication/. [cite. 2024-11-24].

[8] Leo Well, *Add clickable buttons to your Python UI* Available from: https://www.pythonguis.com/docs/qpushbutton/. [cite. 2024-11-30].

[9] doc.qt.io, *Styles and Style Aware Widgets* Available from: https://doc.qt.io/qt-6/style-reference.html. [cite. 2024-11-30].

[10] Martin Fitzpatrick, *Use layouts to effortlessly position widgets within the window* Available from: https://www.pythonguis.com/tutorials/pyqt-layouts/. [cite. 2024-11-30].

[11] Leo Well, *A Simple Text Input Widget* Available from: https://www.pythonguis.com/docs/qlineedit-widget/. [cite. 2024-11-28].

[12] tutorials point, *PyQt5 - Introduction* Available from: https://www.tutorialspoint.com/pyqt5/pyqt5_quick_guide.htm. [cite. 2024-11-28]

[13] doc.qt.io, *QEvent* Available from: https://doc.qt.io/qtforpython-5/PySide2/QtCore/QEvent.html. [cite. 2024-11-28]

[14] doc.qt.io, *QWebChannel* Available from: https://doc.qt.io/qtforpython-5/PySide2/QtWebChannel/QWebChannel.html. [cite. 2024-11-28]

[15] Martin Fitzpatrick, *PyQt5 Signals, Slots & Events* Available from: https://www.pythonguis.com/tutorials/pyqt-signals-slots-events/. [cite. 2024-11-28]

[16] doc.qt.io, *QWebEngineView*. Available from: https://doc.qt.io/qtforpython-5/PySide2/QtWebEngineWidgets/QWebEngineView.html. [cite. 2024-11-29].

[17] Riverbank Computing. *PyQtWebEngine - Python Bindings for the Qt WebEngine Framework*. Available from: https://pypi.org/project/PyQtWebEngine/. [cite. 2024-11-27]

[18] Pypi.org *Request 2.31.0*. Available from: https://pypi.org/project/requests/. [cite. 2024-11-27]

[19] John Millikin. *jsonlib 1.6.1*. Available from: https://pypi.org/project/jsonlib/. [cite. 2024-11-21]

[20] Pratik Choudhari. *How to work with tarball/tar files in Python*. Available from: https://www.python-engineer.com/posts/tarfile-python/. [cite. 2025-03-21]

[21] Shubham. *Python os module*. Available from: https://www.digitalocean.com/community/tutorials/python-os-module. [cite. 2024-11-21]

[22] Michael Driscoll. *The sys Module*. Available from: https://python101.pythonlibrary.org/chapter20_sys.html. [cite. 2024-11-22]

[23] nvie. *times 0.7*. Available from: https://pypi.org/project/times/. [cite. 2024-11-22]

[24] Philipp Acsany. *Dependency Management With Python Poetry*. Available from: https://realpython.com/dependency-management-python-poetry/. [cite. 2025-04-22]

[25] https://doc.qt.io/ *WebEngine Widgets Simple Browser Example*. Available from: https://doc.qt.io/qt-6/

qtwebengine-webenginewidgets-simplebrowser-example.html. [cite. 2024-12-02]

[26] Arnaldo Gunzi. *Keras in a single McCulloch-Pitts neuron.* Available from: https://blog.chatbotslife.com/keras-in-a-single-mcculloch-pitts-neuron-317397cccd45. [cite. 2024-12-02]

[27] Aakash Gupta. *Everything you need to know about AI (for PMs and builders).* Available from: https://www.news.aakashg.com/p/ai-foundations-for-pms. [cite. 2025-03-02]

[28] Ryan Kraus. *What is Machine Learning?.* Available from: https://www.redhat.com/en/blog/what-machine-learning. [cite. 2025-04-12]

[29] Codewave. *History and Development of Neural Networks in AI.* Available from: https://codewave.com/insights/development-of-neural-networks-history/. [cite. 2025-04-14]

[30] The Pandas Development Team. *pandas: powerful Python data analysis toolkit* Available from: https://pypi.org/project/pandas/. [cite. 2025-01-22].

[31] Gael Varoquaux. *Lightweight pipelining with Python functions* Available from: https://pypi.org/project/joblib/. [cite. 2025-02-27].

[32] scikit-learn developers. *A set of python modules for machine learning and data mining* Available from: https://pypi.org/project/scikit-learn/. [cite. 2025-02-27].

Symbols and abbreviations

HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
GUI	Graphical User Interface
URL	Uniform Resource Locator
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
ML	Machine Learning

A Contents of the electronic appendix

To aid orientation within the project files, a color legend has been defined as following:

- Blue: Extension files from previous thesis.
- Red: Newly created files developed specifically within this thesis.
- Black: Files that were adopted or automatically generated without modifications.

```
senior-os/ ..... Directory for the live ISO build
├── sconf/ ..... Main directory of sconf project
│   ├── icons ..... Icons for apps in senior-os
│   └── phish/
│       ├── sweb_allowed_url.txt ..... Blacklist for sweb
│       └── sweb_phish.txt ..... Whitelist for sweb
└── sweb/ ..... Main directory of the senior web browser project
    ├── credits.txt ..... Acknowledgment and contributor credits
    ├── docs/ ..... User manuals in multiple languages
    │   ├── sweb_manual_cz.md ..... Manual in Czech (Markdown and PDF)
    │   ├── sweb_manual_de.md ..... Manual in German (Markdown and PDF)
    │   └── sweb_manual_en.md ..... Manual in English (Markdown and PDF)
    ├── poetry.lock ..... Dependency versions locked by Poetry
    ├── pyproject.toml ..... Project metadata and dependencies
    ├── README.cz.md ..... Czech project description
    ├── README.de.md ..... German project description
    ├── README.md ..... Main English project description
    ├── requirements.txt ..... List of Python packages for manual installation
    ├── screens/ ..... Visual assets for documentation and presentation
    │   ├── sweb_menu1.png ..... Main menu screenshot
    │   ├── sweb_menu2_cz.png ..... Czech localized menu screenshot
    │   ├── sweb_menu2_de.png ..... German localized menu screenshot
    │   ├── sweb_menu2_en.png ..... English localized menu screenshot
    │   ├── sweb_phishing_cz.png ..... Phishing warning in Czech
    │   ├── sweb_phishing_de.png ..... Phishing warning in German
    │   ├── sweb_phishing_diagram.gif ..... Phishing detection flow diagram
    │   ├── sweb_phishing_en.png ..... Phishing warning in English
    │   ├── sweb_screen_1.png ..... Browser UI screenshot 1
    │   ├── sweb_screen_2.png ..... Browser UI screenshot 2
    │   └── sweb_screen_3.png ..... Browser UI screenshot 3
    └── src/sweb/ ..... Source code for the Sweb application
        ├── browser/browser_core.py ..... Core logic for web browsing
        ├── __init__.py ..... Package initializer
        ├── __main__.py ..... Entry point for running Sweb
        ├── language/language_translator.py ..... Text translation for multilingual
        └── ml/ ..... Machine learning-based phishing detection
            └── ml_check_url.py ..... Detection script using trained model
```

```
    ├── phishing_detection_model.pkl ... Trained phishing detection model
    └── tfidf_vectorizer.pkl ..... Vectorizer for ML model input
├── phish/ ..... Phishing notification and update tools
    ├── notification_email.py ..... Sends alert email to guardian
    └── update_phishing.py ..... Updates phishing URL database
├── ui/window.py ..... Graphical user interface definition
├── utils/ ..... Utility scripts
    ├── monitor_provider.py ..... Monitors web activity
    └── url_blocker.py ..... Blocks suspicious or blacklisted URLs
└── sweb_run.py ..... Main executable script to launch the browser
```