

**Uczelnia Techniczno-Handlowa**

**Im. Heleny Chodkowskiej**

**Wydział Inżynieryjny**

**Kierunek Informatyka**

**Rafał Krajewski**

Nr albumu 32843

**Projekt narzędzia do analizy  
phishingu i implementacja w  
technologii JavaScript**

Praca Inżynierska

napisana pod kierunkiem

dr inż. Michała Malinowskiego

Warszawa, luty 2025



# Spis treści

Wstęp.....	5
1    Wprowadzenie do phishing i cyberbezpieczeństwa .....	7
1.1    Phishing .....	7
1.1.1    Definicja phishingu .....	7
1.1.2    Rodzaje phishingu .....	8
1.2    Cyberprzestrzeń i cyberbezpieczeństwo .....	9
1.2.1    Definicja cyberprzestrzeni.....	9
1.2.2    Definicja cyberbezpieczeństwa.....	10
1.2.3    Cyberprzestrzeń jako obszar zagrożeń .....	10
2    Budowa narzędzia do analizy phishing .....	11
2.1    Struktura kodu źródłowego .....	11
2.1.1    Folder img .....	13
2.1.2    Folder node_modules .....	14
2.1.3    Plik carousel.html .....	16
2.1.4    Plik const.js .....	17
2.1.5    Plik nav.html.....	18
2.1.6    Plik ochronaphi.html.....	20
2.1.7    Pliki package-lock.json oraz package.json.....	22
2.1.8    Plik phi.html .....	23
2.1.9    Plik phishingrozpoznanie.js.....	24
2.1.10    Plik rozpoznaniephi.html.....	26
2.1.11    Plik sec.html .....	27
2.1.12    Plik server.js .....	29
2.1.13    Plik skrypty.js .....	35
2.1.14    Plik sprawdzeniephi.html .....	39
2.1.15    Plik sprawdzenieurl.html.....	40
2.1.16    Plik strona.html .....	41
2.1.17    Plik style.css .....	43
2.1.18    Plik urlrozpoznanie.js .....	49
2.2    Instalacja i konfiguracja środowiska .....	51
3    Opis interfejsu użytkownika i testy aplikacji .....	53
3.1    Prezentacja interfejsu użytkownika.....	53
3.1.1    Strona główna (strona.html).....	53
3.1.2    Informacje o phishingu (phi.html).....	54
3.1.3    Jak rozpoznać phishing (rozpoznaniephi.html).....	54
3.1.4    Jak się chronić przed phishingiem (ochronaphi.html).....	55

3.1.5	Analiza treści e-mail (sprawdzeniephi.html) .....	56
3.1.6	Analiza adresu URL (sprawdzenieurl.html) .....	56
3.2	Testy i analiza wyników .....	57
3.2.1	Metodyka testowania .....	57
3.2.2	Scenariusze testowe .....	58
3.3	Analiza wyników testów .....	63
3.4	Ocena wydajności i stabilności .....	65
4	Wnioski i perspektywy rozwoju.....	67
4.1	Podsumowanie pracy.....	67
4.2	Ocena osiągnięcia celów pracy .....	67
4.3	Możliwości dalszego rozwoju systemu .....	68
4.4	Potencjalne zastosowania praktyczne .....	68
5	Bibliografia .....	70
6	Spis rysunków i tabel .....	72

# Wstęp

Niniejsza praca dyplomowa inżynierska poświęcona jest zagadnieniom, które dotyczą cyberataków typu phishing. Obecnie na świecie obserwuje się tendencję wzrostową ataków na: smartfony, komputery stacjonarne, laptopy, tablety, urządzenia IoT[1] czy nawet na specjalistyczne urządzenia np.: systemy monitorujące stan zdrowia pacjenta. Te ataki mają na celu wyłudzić od nas wrażliwe dane, które później mogą zostać sprzedane czy wykorzystane do czynów, które będą sprawiały właścicielom tych danych problemy.

Temat został wybrany ze względu na problemy, które mogą wystąpić w następstwie cyberataku. Ważnym aspektem tych ataków jest to, aby być w stanie odróżniać co jest prawdziwe, a co jest próbą wyłudzenia danych. Wykonana praca oraz przedstawione następnie dokonania, są kierowane do osób, które chcą rozszerzyć swoją wiedzę w zakresie swojego własnego cyberbezpieczeństwa. Rezultaty przedstawione w dalszej części pracy mogą okazać się interesujące dla odbiorców niezależnie od poziomu ich wcześniejszej wiedzy na temat zagadnienia.

Celem niniejszej pracy inżynierskiej jest zaprojektowanie oraz zaimplementowanie narzędzia służącego do analizy potencjalnych ataków phishingowych z wykorzystaniem języka programowania JavaScript. Definicja języka JavaScript została szczegółowo przedstawiona w opracowaniu T.Kozona, który w swoim artykule blogowym omówi jego cechy główne, zastosowania oraz znaczenie w kontekście tworzenia dynamicznych elementów stron internetowych[2]. Aplikacja ma umożliwiać analizę treści wiadomości e-mail oraz adresów URL pod kątem zgodności ze znanymi wzorcami phishingu. Zgodnie z wpisem na blogu Semcore, adres URL (ang. Uniform Resource Locator) to ciąg znaków wpisywany w przeglądarce, który prowadzi użytkownika do określonego zasobu internetowego[3]. Dodatkowo praca ta ma na celu podniesienie poziomu świadomości społeczeństwa w obszarze, który w dalszej części niniejszej pracy dyplomowej inżynierskiej zostanie szczegółowo omówiony. Jednakże aby osoby, dla których jest przeznaczony projekt rozwinęły swoją wiedzę bądź utrwaliły, potrzebna jest chęć zrozumienia jak wyglądają takie cyberataki, jak je odróżniać, co zrobić gdy zostaniemy zaatakowani i wiele innych zmiennych.

Główny problem badawczy niniejszej pracy dotyczy możliwości skutecznego rozpoznawania ataków phishingowych na podstawie analizy treści wiadomości e-mail oraz adresów URL. Zakłada się, że wykorzystując technologię JavaScript oraz metody

przetwarzania języka naturalnego i porównywania domen, możliwe jest opracowanie narzędzia webowego, które pozwoli użytkownikowi zidentyfikować potencjalne zagrożenia phishingowe w sposób zautomatyzowany i przystępny.

Zakres niniejszej pracy obejmuje zaprojektowanie, implementację oraz wstępną ewaluację narzędzia służącego do analizy potencjalnych ataków phishingowych, przeznaczonego dla użytkowników końcowych. Praca koncentruje się na zagadnieniach związanych z cyberbezpieczeństwem, a w szczególności z identyfikacją zagrożeń wynikających z fałszywych wiadomości e-mail i podejrzanych adresów URL.

W ramach realizacji celu pracy zastosowano następującą metodykę:

- przeprowadzono analizę literatury przedmiotu dotyczącej technik phishingu, charakterystyki cyberzagrożeń oraz narzędzi wykrywających ataki socjotechniczne. Zgodnie z raportem NASK (publikowanym przez PIW Parczew), atak socjotechniczny to wyłudzenie poufnych informacji poprzez podszywanie się pod zaufaną osobę lub instytucję[4];
- dokonano identyfikacji cech charakterystycznych dla treści phishingowych – zarówno w kontekście językowym (słowa kluczowe, struktury wiadomości), jak i technologicznym (cechy domen, podejrzane rozszerzenia URL);
- zaprojektowano architekturę systemu oraz wyodrębnilo komponenty funkcjonalne aplikacji;
- Do implementacji wykorzystano HTML (język znaczników służący do strukturyzowania zawartości stron internetowych), CSS (język opisujący wygląd elementów HTML) oraz JavaScript. W części serwerowej zastosowano środowisko Node.js służące do uruchamiania aplikacji[5] oraz bibliotekę natural do analizy tekstu,
- przeprowadzono testy funkcjonalne i analizę działania narzędzia na podstawie zdefiniowanych scenariuszy wykrywania treści phishingowych oraz podejrzanych adresów internetowych.

Metodyka pracy łączy podejście projektowe z elementami badań stosowanych w zakresie inżynierii oprogramowania i bezpieczeństwa systemów informatycznych.

# **1 Wprowadzenie do phishing i cyberbezpieczeństwa**

## **1.1 Phishing**

Współcześnie dostęp do Internetu jest powszechny na całym świecie. Patrząc z tej perspektywy zagrożenia czyhają na nas wszędzie, czy to na portalach społecznościowych, forach internetowych czy nawet w otrzymywanych przez nas mailach lub SMS-ach. W związku z tym konieczne jest zwiększanie świadomości społeczeństwa o zagrożeniach jakie mogą wystąpić, i jakie konsekwencje mogą za sobą nieść musi zostać zwiększona[6].

### **1.1.1 Definicja phishingu**

Phishing definiowany przez Komisję Nadzoru Finansowego jest jako metoda oszustwa, której celem jest wyłudzenie poufnych danych takich jak np.: dane logowania do portali społecznościowych, dane do logowania w bankowości elektronicznej czy nawet danych dowodu osobistego, zgodnie z artykułem Jancelewicz[7]. Osoba, której celem jest wykradnięcie naszych danych podszywa się pod znane nam osoby czy organizacje takie jak np.: bank.

Atak phishing wykorzystuje tak zwaną inżynierię społeczną, czyli technikę, która polega na manipulacji lub nakłonienia potencjalnej ofiary do podania informacji zgodnie z zamierzeniami osoby atakującej. Phishing to szerokie pojęcie, które wymaga bardziej szczegółowej klasyfikacji, aby dokładniej opisać sposób przeprowadzania ataku.

Termin phishing przypomina brzmieniowo angielskie słowo fishing, czyli łowienie ryb – i nie bez powodu. Cyberprzestępcy, podobnie jak wędkarze, wykorzystują specjalnie przygotowaną "przynętę", by zwabić ofiarę.

### 1.1.2 Rodzaje phishingu

Phishing można podzielić na różne kategorie w zależności od takich czynników jak:

- Metoda kontaktu z ofiarą (atakujący korzystają ze wszystkich dostępnych środków komunikacji np.: sms, e-mail, połączenie telefoniczne, wiadomości na komunikatorach internetowych),
- Grupa docelowa (napastnicy atakują osoby fizyczne jak i firmy czy organizacje rządowe i pozarządowe. Ataki dzielą się na dwa rodzaje, czyli dotyczące dużej ilości osób np.:
  - atak na skalę światową w czasie pandemii COVID-19, w której napastnicy wykorzystywali strach i niepewność społeczeństwa tworząc fałszywe strony internetowe i wiadomości związane z pandemią. Zgodnie z artykułem *"Scam Pandemic: How Attackers Exploit Public Fear through Phishing"* autorzy przeprowadzili analizę danych, wskazując na znaczny wzrost liczby ataków phishingowych w okresie pandemii COVID-19[8]., Oraz takie które dotyczą konkretnej osoby, czyli np.:
    - atak polegający na podszywaniu się pod znajomą nam osobę i wysłaniu wiadomości na komunikatorze internetowym, w treści której znajduje się link do fałszywej strony. Ofiara ufająca „znajomemu”, po kliknięciu w odnośnik przechodzi do strony internetowej i wpisuje swoje dane logowanie, które następnie zostają przechwycone przez napastnika. Zgodnie z artykułem Grzbieleca[9].,
- Rodzaj złośliwej treści (rozdzielamy kilka rodzajów ataków phishing):
  - Phishing: metoda oszustwa, której celem jest wyłudzenie poufnych danych od większej grupy osób takich jak np.: dane logowania do portali społecznościowych, dane do logowania w bankowości elektronicznej czy nawet danych dowodu osobistego.,
  - Spear-phishing: metoda ta jest bardziej skoncentrowana na pojedynczej osobie lub małej grupie ofiar np.: „znajomy” wysła wiadomość na komunikatorze internetowym, z potrzebą wysłania mu 20zł blikiem<sup>1</sup>.

---

<sup>1</sup> Blik - system płatności mobilnych w Polsce, uruchomiony 9 lutego 2015, umożliwiający użytkownikom smartfonów płatności bezgotówkowe w sklepach stacjonarnych lub internetowych, wypłacanie i wpłacanie gotówki w bankomatach oraz realizowanie przelewów na numer telefonu, a także generowanie



- Whaling: sposób ten jest taki sam jak spear-phishing, jednakże jest ukierunkowany na pracowników wysokiego szczebla np.: dyrektorzy, doktoranci, politycy itp.,
- Clone-phishing: metoda ta polega jak sama nazwa wskazuje na sklonowaniu oryginalnej wiadomości e-mail i zamienienia linków lub załączników w niej na takie, które są fałszywe np.: zamiana linku logowania do strony operatorów sieci komórkowych, na wyglądające identycznie jednakże zapisujące dane na serwerach oszustów.
- Vishing: sposób ten to phishing telefoniczny, czyli np.: oszust dzwoni do ofiary używając modulatora głosu podszywając się pod osobę nam znaną, prosząc o szybki przelew blik.,
- Smishing: metoda ta to phishing sms-owy, czyli np.: atakujący przesyła wiadomość tekstową (SMS) o treści „PGE: Na dzień 20.03 zaplanowano odłączenie energii elektrycznej! Prosimy o uregulowanie należności 10.50 zł Zapłać teraz na <https://pgezaplata.co/>” ).

## 1.2 Cyberprzestrzeń i cyberbezpieczeństwo

### 1.2.1 Definicja cyberprzestrzeni

Cyberprzestrzeń jest definiowana przez Biuro Bezpieczeństwa Narodowego jako przestrzeń przetwarzania i wymiany informacji tworzona przez systemy teleinformatyczne[10]. Z kolei A.Warchoł w publikacji „Pojęcie cyberprzestrzeni w strategiach bezpieczeństwa państw członkowskich Unii Europejskiej” szczegółowo omawia, jak obszar ten jest traktowany w dokumentach UE[11]. Ze względu na dominującą rolę w nowoczesnych systemach państwowych, społecznych i gospodarczych, cyberprzestrzeń stała się kluczowym wymiarem bezpieczeństwa państwa obok lądu, morza, powietrza i przestrzeni kosmicznej[12].

---

czeków z cyfrowym kodem. Pełna definicja i treść jest dostępna pod adresem:  
[https://pl.wikipedia.org/wiki/Blik\\_\(system\\_p%C5%82atno%C5%9Bci\)](https://pl.wikipedia.org/wiki/Blik_(system_p%C5%82atno%C5%9Bci))

### **1.2.2 Definicja cyberbezpieczeństwa**

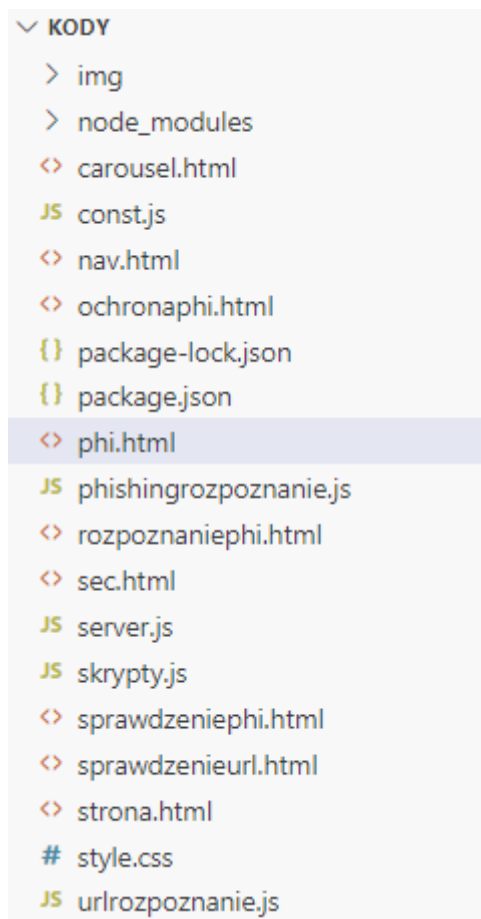
Cyberbezpieczeństwo to odporność systemów informacyjnych na działania naruszające poufność, integralność, dostępność oraz autentyczność przetwarzanych danych lub związanych z nimi usług oferowanych przez te systemy, a także jako działania mające na celu ochronę tych celów. Katarzyna Chałubińska-Jentkiewicz w artykule „Cyberbezpieczeństwo – zagadnienia definicyjne” analizuje szczegółowo te pojęcia, podkreślając znaczenie ich precyzyjnego zdefiniowania z prawnego punktu widzenia[13].

### **1.2.3 Cyberprzestrzeń jako obszar zagrożeń**

W cyberprzestrzeni czyha na nas wiele zagrożeń, takich jak kradzież danych, cyberprzemoc, uzależnienie od Internetu czy niebezpieczne kontakty online. Szczególnie narażeni są młodzi i starsi ludzie, którzy często nieświadomie wchodzi w kontakt ze szkodliwymi treściami, np.: pornografia, przemoc czy fałszywe strony internetowe. Autorzy publikacji zwracają szczególną uwagę, że zagrożenia te są coraz bardziej złożone i często trudno je odróżnić od prawdziwych treści, co utrudnia skuteczną ochronę[14]. Dlatego ważne jest rozwijanie świadomości cyfrowej i odpowiedzialne korzystanie z dostępnej technologii. Z kolei w monografii „Kultura fizyczna i bezpieczeństwo. Wybrane aspekty” podkreślono, że współczesne zagrożenia w cyberprzestrzeni obejmują m.in. cyberterroryzm, który stanowi poważne niebezpieczeństwo dla infrastruktury krytycznej, zwłaszcza w sektorze energetycznym. Rozwój nowoczesnych systemów sterowania i monitorowania zwiększa co prawda efektywność, ale jednocześnie czyni je podatnymi na cyberataki, które mogą zagrażać bezpieczeństwu państwa i obywateli[15].

## 2 Budowa narzędzia do analizy phishing

### 2.1 Struktura kodu źródłowego



Rysunek 1. Struktura kodu źródłowego - Opracowanie: Rafał Krajewski

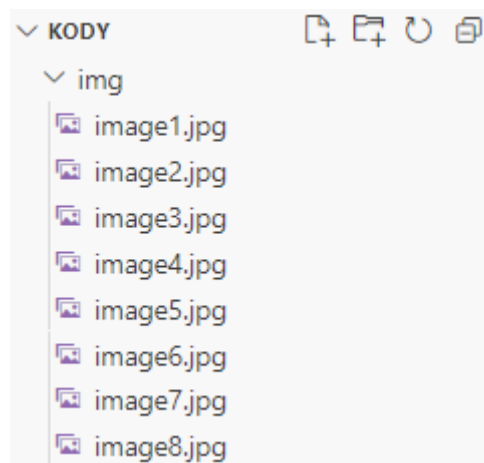
Powyższy rysunek przedstawia strukturę głównego katalogu „KODY”, który zawiera foldery oraz pliki niezbędne do poprawnego funkcjonowania strony internetowej. W skład folderu „KODY” wchodzi:

- Foldery:
  - i. img – folder przechowujący zasoby graficzne,
  - ii. node\_modules – folder automatycznie generowany przez środowisko Node.js zawierający wszystkie zainstalowane moduły.

- Pliki HTML:
  - i. carousel.html – plik zawierający komponent karuzeli wyświetlający zasoby graficzne,
  - ii. nav.html – plik odpowiadający za strukturę nawigacji strony,
  - iii. ochronaphi.html – plik przedstawiający metody ochrony przed phishingiem,
  - iv. phi.html – plik opisujący co to jest phishing oraz objaśnia kilka najczęściej spotykanych typów phishingu,
  - v. rozpoznaniephi.html – plik wyjaśniający użytkownikowi jak rozpoznać czy wiadomość, którą dostaliśmy jest phishingiem,
  - vi. sec.html – plik przedstawiający część struktury strony internetowej, której głównym celem jest umożliwienie wysłania testowego maila phishingowego na wpisany przez użytkownika adres,
  - vii. sprawdzeniephi.html – plik mający na celu sprawdzenie wpisanej treści e-mail przez użytkownika pod kątem phishingu,
  - viii. sprawdzenieurl.html - plik mający na celu analizę wpisanego adresu URL użytkownika,
  - ix. strona.html – główny plik strony internetowej, pełniący funkcję strony startowej.
- Pliki JavaScript:
  - i. const.js – zawiera zmienne z danymi, używane w projekcie,
  - ii. phishingrozpoznanie.js – skrypt odpowiadający za rozpoznanie podejrzanej treści e-mail,
  - iii. urlrozpoznanie.js – skrypt odpowiadający za przetwarzanie adresów URL,
  - iv. server.js – plik pełniący rolę skryptu serwera odpowiedzialnego za backend[16] projektu,
  - v. skrypty.js – plik zawiera dodatkowe funkcjonalności JavaScript wykorzystywane na stronie internetowej.
- Pliki JSON:
  - i. package.json – podstawowy plik konfiguracyjny dla serwera Node.js,

- ii. package-lock.json – odpowiada za dokładne wersje zainstalowanych modułów, zapewniając spójność środowiska.
- Plik CSS:
  - i. style.css – arkusz stylów kaskadowych odpowiedzialny za wygląd strony.

### 2.1.1 Folder img



*Rysunek 2. Zawartość folderu img - Opracowanie: Rafał Krajewski*

Zgodnie z rysunkiem 2 w tym folderze znajdują się wszystkie obrazki, zdjęcia etc. użyte do stworzenia strony internetowej, a dokładniej karuzeli zamieszczonej na stronie głównej projektu.

## 2.1.2 Folder node\_modules

node_modules	node_modules	node_modules
> .bin	> dotenv	> inherits
> @mongodb-js	> dunder-proto	> ipaddr.js
> @redis	> ee-first	> kareem
> @types	> encodeurl	> math-intrinsics
> abort-controller	> es-define-property	> media-typer
> accepts	> es-errors	> memjs
> afinn-165	> es-object-atoms	> memory-pager
> afinn-165-financialmarketnews	> es-set-tostringtag	> merge-descriptors
> agentkeepalive	> escape-html	> methods
> ansi-styles	> etag	> mime
> apparatus	> event-target-shim	> mime-db
> array-flatten	> eventemitter3	> mime-types
> async	> express	> minimalist
> asynckit	> finalhandler	> mongodb
> basic-auth	> follow-redirects	> mongodb-connection-string-url
> body-parser	> form-data	> mongoose
> bson	> form-data-encoder	> mpath
> bytes	> formdata-node	> mquery
> call-bind-apply-helpers	> forwarded	> ms
> call-bound	> fresh	> natural
> chalk	> function-bind	> negotiator
> cluster-key-slot	> generic-pool	> node-domexception
> color-convert	> get-intrinsic	> node-fetch
> color-name	> get-proto	> nodemailer
> combined-stream	> gopd	> object-assign
> content-disposition	> has-flag	> object-inspect
> content-type	> has-symbols	> on-finished
> cookie	> has-tostringtag	> openai
> cookie-signature	> hasown	> opener
> cors	> he	> parseurl
> corser	> html-encoding-sniffer	> path-to-regexp
> debug	> http-errors	> pg
> delayed-stream	> http-proxy	> pg-cloudflare
> depd	> http-server	> pg-connection-string
> destroy	> humanize-ms	> pg-int8
	> iconv-lite	> pg-pool

Rysunek 3. Struktura folderu node\_modules cz.1 - Opracowanie: Rafał Krajewski



Rysunek 4. Struktura folderu node\_modules cz.2 - Opracowanie: Rafał Krajewski

Zgodnie z rysunkami 3 i 4 folder ten zawiera wszystkie moduły, pliki jakie zostały zainstalowane ręcznie oraz te, które są automatycznie dodawane podczas instalacji serwera Node.js.

### 2.1.3 Plik carousel.html

```
<> carousel.html > ...
1  <div class="carousel">
2      <div class="carousel-item active">
3          
4          <div class="carousel-caption">Nigdy nie klikaj w podejrzone linki</div>
5      </div>
6      <div class="carousel-item">
7          
8          <div class="carousel-caption">Zawsze sprawdzaj mail nadawcy</div>
9      </div>
10     <div class="carousel-item">
11         
12         <div class="carousel-caption">Używaj dwu-etapowej weryfikacji</div>
13     </div>
14     <div class="carousel-item">
15         
16         <div class="carousel-caption">Zawsze sprawdzaj adres strony</div>
17     </div>
18     <div class="carousel-item">
19         
20         <div class="carousel-caption">Nie podawaj danych logowania na prośbę e-mailową</div>
21     </div>
22     <div class="carousel-item">
23         
24         <div class="carousel-caption">Nie pobieraj plików z podejrzanych e-maili </div>
25     </div>
26     <div class="carousel-item">
27         
28         <div class="carousel-caption">Nie podawaj wrażliwych danych przez telefon</div>
29     </div>
30     <div class="carousel-item">
31         
32         <div class="carousel-caption">Regularnie aktualizuj oprogramowanie</div>
33     </div>
34 </div>
```

Rysunek 5. Struktura komponentu karuzeli w HTML z komunikatami dotyczącymi cyberbezpieczeństwa – opracowanie: Rafał Krajewski

Plik carousel.html został napisany w języku HTML. Zgodnie z rysunkiem 5 zawiera on znaczniki (<div>)[17], które posiadają klasy[18] dzięki którym kod CSS[19] nadaje style elementom, a JavaScript nimi manipuluje. Na powyższym rysunku widać strukturę pliku oraz jego treść. Klasa „carousel” jest to główna klasa, która zawiera w sobie kolejne elementy typu „carousel-item”. Każdy z tych elementów odpowiada za jeden slajd w karuzeli i zawiera obraz (<img>) oraz podpis (<div class="carousel-caption">), który prezentuje komunikat dotyczący bezpieczeństwa.

Pierwszy slajd jest aktywny domyślnie o czym świadczy klasa active, dzięki czemu jest widoczny od razu po załadowaniu strony. Pozostałe elementy pojawiają się kolejno zgodnie z logiką działania karuzeli, sterowanej przez JavaScript.



## 2.1.4 Plik const.js

```
1  const.js > | cat <unknown> > | phishingExamples
2  module.exports = {
3      phishingExamples: [
4          "Twoje konto zostało zablokowane. Kliknij tutaj, aby przywrócić dostęp.",
5          "Dostęp do konta został ograniczony z powodu podejrzanego aktywności.",
6          "Kliknij link, aby zaktualizować swoje dane logowania.",
7          "Otrzymałeś fakturę. Sprawdź załącznik, aby ją opłacić.",
8          "Twoje konto bankowe zostało zablokowane. Zaloguj się, aby odblokować.",
9          "Nieudana próba logowania. Zweryfikuj swoje dane tutaj.",
10         "Twoja paczka czeka na odbiór. Kliknij, aby potwierdzić.",
11         "Twoje konto e-mail zostanie usunięte. Kliknij, aby zapobiec.",
12         "Otrzymałeś zwrot podatku. Wypełnij formularz, aby odebrać.",
13         "Twoje konto zostało zaatakowane. Zmień hasło natychmiast.",
14         "Twoje dane logowania wygasły. Zaktualizuj je tutaj.",
15         "Otrzymałeś nową wiadomość. Kliknij, aby przeczytać.",
16         "Twoje konto zostało użyte w podejrzanym transakcji. Sprawdź szczegóły.",
17         "Twoje konto PayPal zostało ograniczone. Zweryfikuj tożsamość.",
18         "Otrzymałeś nową fakturę. Kliknij, aby zobaczyć szczegóły.",
19         "Twoje konto zostało zawieszono. Kliknij, aby aktywować.",
20         "Twoje dane osobowe są zagrożone. Kliknij, aby zabezpieczyć.",
21         "Twoje konto zostało oznaczone jako podejrzanego. Zweryfikuj tożsamość.",
22         "Otrzymałeś nową wiadomość głosową. Kliknij, aby odsłuchać.",
23         "Twoje konto zostało zablokowane z powodu nieautoryzowanego dostępu. Kliknij tutaj, aby odblokować.",
24         "Your account has been suspended. Click here to restore access.",
25         "Suspicious activity detected on your account. Verify your details.",
26         "Click the link to update your login credentials.",
27         "You have received an invoice. Check the attachment to pay it.",
28         "Your bank account has been locked. Log in to unlock it.",
29         "Failed login attempt detected. Verify your account here.",
30         "Your package is waiting for pickup. Click to confirm.",
31         "Your email account will be deleted. Click to prevent this.",
32         "You have received a tax refund. Fill out the form to claim it.",
33         "Your account has been compromised. Change your password immediately.",
34         "Your login credentials have expired. Update them here.",
35         "You have received a new message. Click to read it.",
36         "Your account was used in a suspicious transaction. Check details.",
37         "Your PayPal account has been limited. Verify your identity.",
38         "You have received a new invoice. Click to view details.",
39         "Your account has been suspended. Click to activate it.",
40         "Your personal data is at risk. Click to secure it.",
41         "Your account has been flagged as suspicious. Verify your identity.",
42         "You have received a new voicemail. Click to listen.",
43         "Your account has been locked due to unauthorized access. Click here to unlock."
44     ],
45     suspiciousTLDs: [
46         ".ru", ".tk", ".ml", ".gq", ".cf", ".biz", ".info", ".xyz", ".top",
47         ".click", ".loan", ".win", ".party", ".cam", ".date", ".men", ".stream",
48         ".icu", ".pw", ".work", ".host", ".space", ".website", ".online", ".shop",
49         ".fun", ".pro", ".vip", ".club", ".site", ".tech", ".store", ".press",
50         ".live", ".center", ".trade", ".rocks", ".guru", ".solutions"
51     ],
52     knownLegitDomains: [
53         "paypal.com", "microsoft.com", "apple.com", "google.com", "facebook.com",
54         "amazon.com", "netflix.com", "bankofamerica.com", "chase.com", "google.pl",
55         "onet.pl", "wp.pl", "interia.pl", "allegro.pl", "olx.pl",
56         "gov.pl", "poczta-polska.pl", "mbank.pl", "ing.pl", "millennium.pl",
57         "santander.pl", "pekao24.pl", "pzu.pl", "zus.pl", "tvn.pl",
58         "tvp.pl", "gazeta.pl", "wyborcza.pl", "rmf.fm", "radiozet.pl",
59         "empik.com", "medonet.pl", "o2.pl", "play.pl", "plus.pl",
60         "orange.pl", "tmobile.pl", "cyfrowypolsat.pl", "biedronka.pl", "lidl.pl"
61     ]
62 }
63 ;
```

Rysunek 6. Struktura pliku ze zmiennymi w JavaScript - Opracowanie: Rafał Krajewski

Plik const.js został napisany w języku JavaScript. Zgodnie z rysunkiem 6 zawiera on zestaw stałych danych, które mogą być wykorzystywane przez inne moduły aplikacji

w celu wykrywania potencjalnych zagrożeń związanych z phishingiem. Zmienna eksportowana za pomocą „module.exports” zawiera trzy główne tablice danych:

- phishingExamples: tablica zawierająca przykładowe komunikaty, które mogą sugerować próbę phishingu. Wiadomości te naśladują typowe treści wykorzystywane w oszustwach internetowych, np. komunikaty o zablokowanym koncie, prośby o aktualizację danych logowania czy informacje o nieopłaconych fakturach,
- suspiciousTLDs: tablica zawierająca nazwy podejrzanych domen, które są często wykorzystywane w adresach stron stosowanych w atakach phishingowych. W zestawieniu znajdują się rozszerzenia takie jak .ru, .tk, .ml, .gq, .cf, a także nowe i mniej popularne końcówki jak .win, .click czy .stream,
- knownLegitDomains: tablica zawierająca listę znanych i zaufanych domen należących do dużych, renomowanych firm. Służy ona do porównywania i identyfikowania legalnych adresów internetowych w procesie walidacji.

Struktura pliku została przygotowana w sposób umożliwiający łatwą integrację z modułami aplikacji analizującej treści e-maili oraz adresów URL. Dzięki temu możliwe jest szybkie filtrowanie podejrzanych treści i ostrzeżenie użytkownika przed potencjalnym zagrożeniem.

### 2.1.5 Plik nav.html

```
<> nav.html > nav
1  <nav>
2    <ul>
3      <a href="strona.html" class="home-button">
4        <i class="fas fa-home"></i>
5      </a>
6      <li><a href="phi.html">Co to jest Phishing?</a></li>
7      <li><a href="rozpoznaniephi.html">Jak rozpoznać phishing?</a></li>
8      <li><a href="ochronaphi.html">Jak się chronić przed phishingiem?</a></li>
9      <li><a href="sprawdzeniephi.html">Sprawdź czy dostałeś podejrzaną wiadomość</a></li>
10     <li><a href="sprawdzenieurl.html">Sprawdź czy adres URL jest podejrzany</a></li>
11   </ul>
12 </nav>
```

Rysunek 7. Struktura pliku nawigacji w HTML - Opracowanie: Rafał Krajewski

Plik nav.html został napisany w języku HTML. Zgodnie z rysunkiem 7 zawiera on listę nieuporządkowaną (<ul>), w której znajdują się hiperłącza (<a>) do poszczególnych podstron serwisu internetowego. Struktura ta jest umieszczona wewnątrz elementu (<nav>), odpowiadającego za sekcje nawigacyjną na stronie internetowej.

Pierwszy element z listy (<li>) zawiera link do strony głównej „strona.html” i jest on oznaczony klasą „home-button”. Wewnątrz tego linku znajduje się ikona domku (<i class=„fas fa-home”>), która pochodzi z biblioteki Font Awesome[20] i pełni funkcję wizualnego oznaczenia przycisku powrotu do strony głównej.

Pozostałe elementy listy zawierają odnośniki prowadzące do podstron związanych z tematyką phishingu, takich jak:

- Wyjaśnienie pojęcia phishingu oraz krótki opis najbardziej popularnych typów phishingu („phi.html”),
- Sposoby jego rozpoznania („rozpoznaniephi.html”),
- Metody ochrony przed tego typu zagrożeniami („ochronaphi.html”),
- Możliwość sprawdzenia podejrzanej wiadomości e-mail („sprawdzeniephi.html”),
- Oraz weryfikacja czy wpisany przez nas adres URL może być podejrany („sprawdzenieurl.html”).

Powyższa struktura jest powszechnie stosowana w projektowaniu stron internetowych jako główne menu nawigacyjne, które pozwala użytkownikowi na łatwe przemieszczanie się między sekcjami witryny.

## 2.1.6 Plik ochronaphi.html

```
1 <!DOCTYPE html>
2 <html lang="pl">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Praca Inżynierska</title>
7   <link rel="stylesheet" href="style.css">
8   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css">
9   <script src="skrypty.js" defer></script>
10 </head>
11 <body>
12   <header>
13     <h1>Projekt narzędzia do analizy phishingu i implementacja w technologii JavaScript</h1>
14     <div id="nav-placeholder"></div>
15   </header>
16   <main>
17     <section id="rozpoznanie">
18       <h2>Jak się chronić przed Phishingiem?</h2>
19       <p>Phishing to metoda oszustwa, w której cyberprzestępcy podszywają się pod zaufane instytucje lub osoby, aby wyłudzić poufne informacje, takie jak hasła, numery kart kredytowych czy dane osobowe. Aby skutecznie chronić się przed phishingiem, warto przestrzegać kilku podstawowych zasad:</p>
20       <ol>
21         <li><strong>Uważaj na podejrzane e-maile i wiadomości:</strong> Nie otwieraj załączników ani nie klikaj w linki w wiadomościach od nieznanych nadawców. Zawsze sprawdzaj adres e-mail nadawcy i zwracaj uwagę na literówki oraz dziwne domeny.</li>
22         <li><strong>Sprawdź adresy URL:</strong> Przed wprowadzeniem danych na stronie internetowej upewnij się, że adres URL zaczyna się od "https://" i zawiera poprawną nazwę domeny. Unikaj klikania w linki w wiadomościach e-mail, zamiast tego wpiszuj adresy ręcznie w przeglądarce.</li>
23         <li><strong>Aktualizuj oprogramowanie:</strong> Regularnie aktualizuj system operacyjny, przeglądarkę internetową oraz oprogramowanie antywirusowe, aby chronić się przed najnowszymi zagrożeniami.</li>
24         <li><strong>Używaj silnych haseł:</strong> Twórz unikalne i skomplikowane hasła dla różnych kont. Unikaj używania tych samych haseł w wielu miejscach. Rozważ korzystanie z menedżera haseł.</li>
25         <li><strong>Włącz uwierzytelnianie dwuskładnikowe (2FA):</strong> Uwierzytelnianie dwuskładnikowe dodaje dodatkową warstwę zabezpieczeń, wymagając drugiego elementu weryfikacji, takiego jak kod SMS lub aplikacja uwierzytelniająca.</li>
26         <li><strong>Szkol się i edukuj:</strong> Regularnie ucz się o nowych zagrożeniach i metodach phishingu. Informuj swoich znajomych i rodzinę o zagrożeniach i sposobach ochrony.</li>
27       </ol>
28       <p>Więcej informacji na temat ochrony przed phishingiem można znaleźć na stronach:</p>
29       <ul>
30         <li><a href="https://cert.pl/szukaj/#phishing" target="_blank">CERT Polska</a></li>
31         <li><a href="https://www.gov.pl/web/gov/szukaj?query=phishing" target="_blank">Ministerstwo Cyfryzacji</a></li>
32         <li><a href="https://www.us-cert.gov/ncas/tips/ST04-014" target="_blank">US-CERT</a></li>
33       </ul>
34     </section>
35   </main>
36   <footer>
37     <p>&copy; 2025 Praca Inżynierska - Rafał Krajewski</p>
38   </footer>
39 </body>
40 </html>
```

Rysunek 8. Struktura pliku strony internetowej o ochronie przed phishingiem w HTML - Opracowanie: Rafał Krajewski

Plik ochronaphi.html został napisany w języku HTML. Zgodnie z rysunkiem 8 zawiera on strukturę strony internetowej, której celem jest przedstawienie metod ochrony przed phishingiem. Kod został podzielony na trzy główne sekcje: <head>, <body> i <footer>. Plik zapisano z uwzględnieniem zasad poprawnej semantyki[21] oraz responsywności[22] poprzez zastosowanie metatagów[23] w sekcji (<head>), m.in. (<meta name="viewport" content="width=device-width, initial-scale=1.0">), co zapewnia prawidłowe wyświetlanie strony na różnych urządzeniach.

W nagłówku (<header>) umieszczono tytuł projektu: „Projekt narzędzia do analizy phishing i implementacja w technologii JavaScript”, a także znacznik (<div>) z identyfikatorem „nav-placeholder”, który służy jako miejsce załadowania menu nawigacyjnego.

Główna część strony została umieszczona w znaczniku (<main>), a w sekcji (<section id="rozpoznanie">) znajduje się podtytuł „Jak chronić się przed phishingiem?” oraz akapit (<p>) wprowadzający, który definiuje zjawisko phishing jako metodę

oszustwa polegającą na podszywaniu się pod zaufane instytucje w celu wyłudzenia poufnych danych użytkownika.

W celu przedstawienia praktycznych wskazówek dotyczących ochrony przed potencjalnymi zagrożeniami, zamieszczono listę uporządkowaną (<ol>), w której każde z podanych zasad zostało umieszczone w oddzielnych elementach listy. Za pomocą tagu (<strong>) wyróżniono kluczowe fragmenty wskazując m.in. na konieczność ostrożności przy otwieraniu wiadomości e-mail, weryfikację adresów URL, aktualizację oprogramowania, stosowanie silnych haseł, uwierzytelnianie dwuskładnikowe oraz edukację w zakresie cyberbezpieczeństwa.

Dodatkowo na stronie zostały zamieszczone odnośniki do źródeł zewnętrznych, które zawierają bardziej szczegółowe i aktualne informacje na temat ochrony przed phishingiem. Hiperłącza te zostały osadzone w oddzielnej liście i kierują m.in. do witryn CERT Polska, Ministerstwa Cyfryzacji oraz US-CERT. Wszystkie odnośniki otwierają się w nowej karcie przeglądarki dzięki zastosowaniu atrybutu (target="\_blank")

Na końcu strony znajduje się stopka (<footer>), zawierająca podpis autora projektu wraz z datą realizacji: „2025 Praca Inżynierska – Rafał Krajewski”. Całość kodu została wzbogacona o odwołania do zewnętrznych arkuszy stylów CSS (<link rel="stylesheet" href="style.css">) oraz skryptów (<script src="skrypty.js" defer>), w tym biblioteki ikon Font Awesome (<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css">) oraz lokalnego pliku „skrypty.js”, co sugeruje, że funkcjonalność strony jest dodatkowo wspierana przez skrypty JavaScript. Atrybut „defer” użyty przy odwołaniu do skryptu, oznacza to, że plik jest ładowany równocześnie ze stroną HTML, jednakże jest wykonywany dopiero po załadowaniu wszystkich plików witryny.

## 2.1.7 Pliki package-lock.json oraz package.json

```
{ } package.json > ...
1  {
2    "dependencies": {
3      "body-parser": "^2.2.0",
4      "cors": "^2.8.5",
5      "express": "^4.21.2",
6      "natural": "^8.1.0",
7      "nodemailer": "^6.10.1",
8      "openai": "^4.104.0",
9      "string-similarity": "^4.0.4"
10   },
11   "name": "kody",
12   "version": "1.0.0",
13   "main": "server.js",
14   "scripts": {
15     "test": "echo \"Error: no test specified\" && exit 1",
16     "start": "node server.js"
17   },
18   "keywords": [],
19   "author": "",
20   "license": "ISC",
21   "description": ""
22 }
```

Rysunek 9. Struktura pliku konfiguracyjnego serwera node.js - Opracowanie: Rafał Krajewski

Zgodnie z rysunkiem 9 pliki te są integralnymi elementami środowiska „node.js” i wykorzystywane są do zarządzania zależnościami w projekcie, który opiera się na ekosystemie JavaScript, a zwłaszcza w menadżerze pakietów npm (Node Package Manager).

Plik package.json zawiera podstawowe informacje konfiguracyjne dotyczące projektu, takie jak jego nazwa, wersja, autor, licencja, a także lista zależności, czyli bibliotek i modułów zewnętrznych, które są wymagane do poprawnego działania aplikacji. W strukturze tego pliku znajdują się m.in. pola takie jak:

- dependencies: biblioteki wymagane w środowisku produkcyjnym,
- name: identyfikator projektu,
- version: aktualna wersja aplikacji,
- main: plik główny,
- scripts: zestaw poleceń umożliwiających uruchamianie zdefiniowanych zadań (np. testów),
- description: krótki opis funkcjonalności.

Jest on kluczowy przy instalacji nowych pakietów przy pomocy komendy „npm install”.

Plik package-lock.json jest automatycznie generowany przez „npm” przy każdej instalacji pakietów. Jego zadaniem jest precyzyjne zapisanie wersji każdej zainstalowanej zależności oraz jej podzależności. Dzięki temu możliwe jest zachowanie spójności środowiska pomiędzy różnymi stanowiskami pracy lub podczas wdrażania aplikacji na różnych serwerach.

## 2.1.8 Plik phi.html

```
1 <!DOCTYPE html>
2 <html lang="pl">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Praca Inżynierska</title>
7   <link rel="stylesheet" href="style.css">
8   <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css">
9   <script src="skrypty.js" defer></script>
10 </head>
11 <body>
12   <header>
13     <h1>Projekt narzędzia do analizy phishingu i implementacja w technologii JavaScript</h1>
14     <div id="nav-placeholder"></div>
15   </header>
16   <main>
17     <section id="phi">
18       <h2>Co to jest Phishing</h2>
19       <p>Phishing to metoda oszustwa internetowego, polegająca na wyłudzeniu poufnych informacji, takich jak hasła czy numery kart kredytowych. Atakujący podszywa się pod zaufane instytucje, takie jak banki, serwisy społecznościowe czy sklepy internetowe. W celu oszukania ofiary, wysyła ona wiadomość e-mail, SMS lub dzwoni podając się za pracownika instytucji. W treści wiadomości znajduje się link do fałszywej strony, na której ofiara ma podać swoje dane. W ten sposób atakujący uzyskuje dostęp do poufnych informacji, które następnie może wykorzystać w celach przestępczych.</p>
20       <h3>Typy phishingu</h3>
21       <ul>
22         <li><strong>Spear phishing</strong> - atak skierowany na konkretną osobę lub organizację, często z wykorzystaniem informacji z mediów społecznościowych.</li>
23         <li><strong>Whaling</strong> - atak na osoby na wysokich stanowiskach, takie jak dyrektorzy czy menedżerowie.</li>
24         <li><strong>Clone phishing</strong> - atak polegający na sklonowaniu oryginalnej wiadomości e-mail i zastąpieniu linków lub załączników złośliwymi.</li>
25         <li><strong>Vishing</strong> - phishing telefoniczny, w którym atakujący dzwoni do ofiary, podszywając się pod zaufaną instytucję.</li>
26         <li><strong>Smishing</strong> - phishing za pomocą wiadomości SMS, w których atakujący podszywa się pod zaufaną instytucję.</li>
27       </ul>
28     </section>
29   </main>
30   <footer>
31     <p>&copy; 2025 Praca Inżynierska - Rafał Krajewski</p>
32   </footer>
33 </body>
34 </html>
```

Rysunek 10. Struktura pliku strony internetowej o rodzajach phishingu w HTML - Opracowanie: Rafał Krajewski

Plik zapisano w języku HTML. Zgodnie z rysunkiem 10 zawiera on strukturę strony internetowej, która jest identyczna jak pliku „ochronaphi.html”, jednakże różni się ona tym co się znajduje w znaczniku (<main>). Witryna ta ma na celu wyjaśnienie co to jest phishing oraz objaśnienie kilku najczęściej spotykanych typów phishingu.

Główna część strony została umieszczona w znaczniku (<main>), a w sekcji (<section id="phi">) znajduje się podtytuł „Co to jest phishing?” oraz akapit (<p>), który w szerszym zakresie opisuje co to jest phishing w porównaniu do witryny „ochronaphi.html”.

W celu przedstawienia typów phishingu, zamieszczono listę nieuporządkowaną (<ul>), w której każde z podanych rodzajów phishingu zostało umieszczone w oddzielnych elementach listy. Za pomocą tagu (<strong>) wyróżniono kluczowe fragmenty wskazując na nazwę typu phishingu.

### 2.1.9 Plik phishingrozpoznanie.js

```
JS phishingrozpoznanie.js > ...
1  const form = document.getElementById("phishing-form");
2  const resultDiv = document.getElementById("result");
3
4  form.addEventListener("submit", async function (e) {
5      e.preventDefault();
6      const input = document.getElementById("email-content").value;
7
8      resultDiv.innerHTML = "🔍 Analizuję treść...<br>";
9
10     const emailResult = await isSimilarToPhishing(input);
11
12
13     const emailSimilarityPercent = (emailResult.similarity * 100).toFixed(2);
14     if (emailResult.similarity > 0.63) {
15         resultDiv.innerHTML += `⚠️ <span style="color:red;">Treść wygląda podejrzanie</span><br>`;
16     } else {
17         resultDiv.innerHTML += `✅ <span style="color:green;">Treść wygląda bezpiecznie</span><br>`;
18     }
19     resultDiv.innerHTML += `📊 Podobieństwo treści e-maila: ${emailSimilarityPercent}%<br><br>`;
20
21
22 });
23
24 async function isSimilarToPhishing(text) {
25     try {
26         const response = await fetch("http://localhost:3000/check-similarity", {
27             method: "POST",
28             headers: { "Content-Type": "application/json" },
29             body: JSON.stringify({ emailText: text })
30         });
31         return await response.json();
32     } catch (error) {
33         resultDiv.innerHTML += "<span style='color:orange;'>❌ Błąd połączenia z serwerem.</span><br>";
34         console.error(error);
35         return { similarity: 0 };
36     }
37 }
38
```

Rysunek 11. Struktura pliku analizującego treść e-maila w JavaScript – Opracowanie: Rafał Krajewski

Zgodnie z rysunkiem 11 plik zapisano w języku JavaScript. Odpowiada on za obsługę formularza dodanego do strony internetowej, który służy do analizy treści wiadomości e-mail pod kątem podobieństwa do znanych przykładów phishingu. Mechanizm ten umożliwia wykrywanie potencjalnie niebezpiecznych wiadomości



poprzez porównanie ich zawartości z uprzednio zidentyfikowanymi wzorcami treści e-maili phishingowych.

W pierwszych dwóch liniach skryptu, są zdefiniowane dwie zmienne „const”, które służą do pobierania elementów HTML ze strony internetowej na podstawie ich identyfikatorów „id”.

Pierwsza zmienna (`const form = document.getElementById(„phishing-form”)`) odwołuje się do formularza (`<form>`) o identyfikatorze „phishing-form”. Jest to formularz, w którym umożliwia się wpisanie przez użytkownika podejrzaną treść e-maila do analizy. Zmienna „form” służy następnie do przypisania obsługi zdarzenia (w tym przypadku „submit”), które wywołuje funkcję sprawdzającą treść e-maila pod kątem phishingu.

Druga zmienna (`const resultDiv = document.getElementById(„result”)`) wskazuje na element (`<div>`), którego identyfikator to „result”. Jest to miejsce, w którym będą wyświetlane wyniki analizy.

Gdy zostanie wprowadzona treść podejrzanego e-maila do pola tekstowego oraz zostanie ona potwierdzona wciśnięciem przycisku „submit”, wykona się funkcja asynchroniczna[24], która przesyła dane do lokalnego serwera za pomocą zapytania typu POST. Kluczowym elementem tej funkcji jest „e.preventDefault()”, która uniemożliwia domyślne odświeżanie strony, co pozwala zachować płynność działania interfejsu i umożliwia wykonanie dalszej logiki kodu bez przerw. Treść wiadomości przekazywana jest w formacie JSON do endpointu[25] „http://localhost:3000/check-similarity”, gdzie zostaje poddana analizie podobieństwa.

W odpowiedzi serwer zwraca obiekt, który zawiera wartość procentową określającą stopień podobieństwa do wiadomości phishingowych. Dzięki temu użytkownik otrzymuje informację zwrotną. W przypadku, gdy wynik przekroczy ustalony próg (w tym przypadku 63%), wyświetla się komunikat ostrzegający przed potencjalnym zagrożeniem. W przeciwnym razie otrzymujemy informację, że treść wygląda bezpiecznie. W dodatku, użytkownik jest informowany o dokładnym poziomie podobieństwa wyrażanym w procentach.

W przypadku gdy nie ma połączenia z serwerem, skrypt ten obsługuje wyjątek, informując o błędzie oraz rejestrując szczegóły problemu w konsoli przeglądarki.

Zapewnia to odporność systemu na błędy związane z brakiem dostępu do usługi backendowej.

Struktura pliku została zaprojektowana w sposób umożliwiający łatwą integrację z aplikacją analizującą treści e-maili. Dzięki takiemu podejściu możliwe jest szybkie filtrowanie treści podejrzanych oraz przekazanie użytkownikowi przejrzystych informacji o potencjalnym zagrożeniu.

### 2.1.10 Plik rozpoznaniephi.html

```
<> rozpoznaniephi.html > html > body > main > section#rozpoznanie > ul > li
1  <!DOCTYPE html>
2  <html lang="pl">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Praca Inżynierska</title>
7      <link rel="stylesheet" href="style.css">
8      <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css">
9      <script src="skrypty.js" defer></script>
10 </head>
11 <body>
12     <header>
13         <h1>Projekt narzędzia do analizy phishingu i implementacja w technologii JavaScript</h1>
14         <div id="nav-placeholder"></div>
15     </header>
16     <main>
17         <section id="rozpoznanie">
18             <h2>Jak rozpoznać phishing?</h2>
19             <p>Phishing jest coraz bardziej zaawansowany i trudny do wykrycia. Jednak istnieje kilka wskazówek,
20             które mogą pomóc rozpoznać oszustwo:</p>
21             <ul>
22                 <li><strong>Sprawdź adres e-mail nadawcy</strong> - zwróć uwagę na domenę, czy nie zawiera ona
23                 literówek ani dodatkowych znaków.</li>
24                 <li><strong>Nie klikaj w podejrzane linki</strong> - zawsze sprawdź adres strony,
25                 na którą prowadzi link.</li>
26                 <li><strong>Uważaj na prośby o podanie poufnych informacji</strong> - żadna instytucja nie prosi
27                 o podanie hasła czy numeru karty kredytowej drogą e-mailową.</li>
28                 <li><strong>Sprawdź treść wiadomości</strong> - zwróć uwagę na błędy ortograficzne,
29                 niechlujne formatowanie czy podejrzane załączniki.</li>
30                 <li><strong>Skontaktuj się bezpośrednio z instytucją</strong> - jeśli masz wątpliwości co do wiadomości,
31                 skontaktuj się bezpośrednio z instytucją, korzystając z oficjalnych danych kontaktowych.</li>
32             </ul>
33         </section>
34     </main>
35     <footer>
36         <p>&copy; 2025 Praca Inżynierska - Rafał Krajewski</p>
37     </footer>
38 </body>
39 </html>
```

Rysunek 12. Struktura strony internetowej o rozpoznawaniu phishingu w HTML – Opracowanie: Rafał Krajewski

Plik zapisano w języku HTML. Zgodnie z rysunkiem 12 zawiera on strukturę strony internetowej, która jest identyczna jak pliku „ochronaphi.html” oraz „phi.html”, jednakże różni się ona tym co się znajduje w znaczniku (<main>). Witryna ta ma na celu wyjaśnienie użytkownikowi jak rozpoznać czy wiadomość, którą dostaliśmy jest phishingiem.

Główna część strony została umieszczona w znaczniku (<main>), a w sekcji (<section id="rozpoznanie">) znajduje się podtytuł „Jak rozpoznać phishing?” oraz akapit (<p>), w którym znajduje się krótkie wprowadzenie do wyżej wymienionego tematu.

W celu przedstawienia kilku możliwości jak rozpoznać phishing, zamieszczono listę nieuporządkowaną (<ul>), w której każde z podanych możliwości wykrzyca phishingu zostało umieszczone w oddzielnych elementach listy. Za pomocą tagu (<strong>) wyróżniono kluczowe fragmenty wskazując na porady dotyczące rozpoznania phishingu.

### 2.1.11 Plik sec.html

```
<> sec.html > section#home > form#phishing-mail > input#email
1  <section id="home">
2    <h2>Zacznijmy od przykładu...</h2>
3    <p>Wpisz poniżej swojego maila, a wyśle do Ciebie przykładowy mail phishing. </p>
4    <form id="phishing-mail">
5      <input type="email" id="email" placeholder="Wpisz swój e-mail" required>
6      <button type="submit" id="mailbtn">Wyślij testowy phishing</button>
7    </form>
8  </section>
9  <section id="about">
10   <h2>Kilka słów o mnie</h2>
11   <p>Mam na imię Rafał. Mam 24 lata i jestem pasjonatem motoryzacji, entuzjastą filmów i seriali oraz
12   początkującym twórcą stron internetowych. Uwielbiam zgłębiać świat samochodów, zarówno tych klasycznych,
13   jak i najnowszych technologii motoryzacyjnych. W wolnym czasie pochłaniamy mnie filmy i seriale,
14   szczególnie te z wciągającą fabułą i ciekawie skonstruowanymi postaciami. Oprócz tego rozwijam swoje
15   umiejętności w tworzeniu stron internetowych, łącząc kreatywność z funkcjonalnością,
16   by dostarczać nowoczesne i estetyczne projekty.</p>
17 </section>
18 <section id="contact">
19   <h2>Kontakt</h2>
20   <li><a href="https://www.linkedin.com/in/rafał-krajewski-880460263" target="_blank">Link do LinkedIn</a></li>
21   <li><a href="mailto:r.krajewski26@gmail.com">Mail do mnie</a></li>
22 </section>
```

Rysunek 13. Struktura sekcji ze strony startowej w HTML - Opracowanie : Rafał Krajewski

Plik zapisano w języku HTML. Zgodnie z rysunkiem 13 zawiera on część struktury strony internetowej, której celem jest umożliwienie użytkownikowi przetestowania, jak może wyglądać przykładowy e-mail phishingowy. Witryna składa się z trzech głównych sekcji, które są osadzone wewnątrz elementu (<main>): „home”, „about”, „contact”.

W sekcji o identyfikatorze „home” umieszczono nagłówek poziomu drugiego (<h2>), który zawiera tekst „Zaczniemy od przykładu...”, a także akapit wyjaśniający funkcjonalność tej sekcji na stronie internetowej. Kluczowym elementem tej sekcji jest formularz (<form>), którego zadaniem jest pobranie od użytkownika adresu e-mail, poprzez wpisanie go do pola e-mailowego, czyli takiego w którym przeglądarka automatycznie sprawdza czy e-mail ma typowe dla niego cechy np.: znak „@”. W tym formularzu znajduje się również przycisk „submit”, który służy do wysłania danych. Użyty atrybut „required” powoduje to, że formularz nie zostanie wysłany bez uprzedniego uzupełnienia adresu e-mail.

W sekcji o identyfikatorze „about” umieszczono nagłówek poziomu drugiego (<h2>), który zawiera tekst „Kilka słów o mnie”, a także akapit, w którym przedstawiono zainteresowania autora oraz jego motywacje związane z tworzeniem stron internetowych. Opis ten pełni funkcję przedstawienia osoby, która jest odpowiedzialna za stworzenie witryny, jednocześnie podkreślając jej umiejętności techniczne oraz zainteresowania.

W sekcji o identyfikatorze „contact”, umieszczono nagłówek poziomu drugiego (<h2>), który zawiera tekst „Kontakt”, a także listę (<li>), w której umieszczono dwa elementy:

- Link do profilu LinkedIn autora
- E-mail do autora strony internetowej

W przypadku odnośnika do LinkedIn autora, zastosowano atrybut „target=„\_blank””, co powoduje otwarcie linku w nowej karcie przeglądarki.

Cała struktura wyżej wymienionych sekcji została zaprojektowana w sposób przejrzysty i logiczny, z zachowaniem odpowiedniej semantyki znaczników, co wpływa pozytywnie na czytelność kodu oraz dostępność strony.

## 2.1.12 Plik server.js

```
JS server.js > app.post("/send-email") callback
1  const express = require("express");
2  const cors = require("cors");
3  const bodyParser = require("body-parser");
4  const natural = require("natural");
5  const nodemailer = require("nodemailer");
6  const { spawn } = require("child_process");
7  const { phishingExamples, suspiciousTLDs, knownLegitDomains } = require("./const");
8
9
10 const app = express();
11 const port = 3000;
12
13 app.use(cors());
14 app.use(bodyParser.json());
15
16 async function generateEmail(prompt) {
17     return new Promise((resolve) => {
18         const ollama = spawn("ollama", ["run", "mistral"]);
19         let output = "";
20
21         ollama.stdin.write(prompt + "\n");
22         ollama.stdin.end();
23
24         ollama.stdout.on("data", (data) => {
25             output += data.toString();
26         });
27
28         ollama.on("close", () => {
29             resolve(output.trim());
30         });
31
32         ollama.on("error", (err) => {
33             console.error("✗ Błąd generowania e-maila:", err);
34             const random = fakeEmails[Math.floor(Math.random() * fakeEmails.length)];
35             resolve(random);
36         });
37     });
38 }
39
40 const transporter = nodemailer.createTransport({
41     service: "gmail",
42     auth: {
43         user: "militox258@gmail.com",
44         pass: "hcnv vjdl gwes mpqw"
45     }
46 });
47
48 app.post("/send-email", async (req, res) => {
49     const { email } = req.body;
50
51     if (!email) {
52         return res.status(400).json({ message: "Brak adresu e-mail!" });
53     }
54 }
```

Rysunek 14. Struktura pliku konfiguracyjnego serwera w JavaScript cz.1 - Opracowanie: Rafał Krajewski

```

JS server.js > app.post("/send-email") callback
48 app.post("/send-email", async (req, res) => {
53   }
54
55   const message = await generateEmail(
56     "Napisz przekonujący, realistyczny e-mail phishingowy, który wygląda jakby był wysłany
57   );
58
59   const mailOptions = {
60     from: "noreply@phimail.com",
61     to: email,
62     subject: "Ważna wiadomość",
63     text: message
64   };
65
66   try {
67     await transporter.sendMail(mailOptions);
68     res.json({ message: "✅ E-mail został wysłany!" });
69   } catch (error) {
70     console.error("❌ Błąd wysyłania e-maila:", error);
71     res.status(500).json({ message: "Wystąpił błąd podczas wysyłania phishingu." });
72   }
73 });
74
75 app.post("/check-similarity", (req, res) => {
76   const { emailText } = req.body;
77
78   if (!emailText) {
79     return res.status(400).json({ error: "Brak tekstu e-maila." });
80   }
81
82   try {
83     const tokenizer = new natural.WordTokenizer();
84     const inputTokens = tokenizer.tokenize(emailText.toLowerCase()).join(" ");
85
86     let maxSimilarity = 0;
87
88     phishingExamples.forEach(example => {
89       const tfidf = new natural.Tfidf();
90       tfidf.addDocument(example.toLowerCase());
91       tfidf.addDocument(inputTokens);
92
93       const vec1 = tfidf.listTerms(0).map(t => t.tfidf);
94       const vec2 = tfidf.listTerms(1).map(t => t.tfidf);
95
96       const length = Math.max(vec1.length, vec2.length);
97       while (vec1.length < length) vec1.push(0);
98       while (vec2.length < length) vec2.push(0);
99
100      const dot = vec1.reduce((sum, v, i) => sum + v * vec2[i], 0);
101      const mag1 = Math.sqrt(vec1.reduce((sum, v) => sum + v * v, 0));
102      const mag2 = Math.sqrt(vec2.reduce((sum, v) => sum + v * v, 0));
103      const similarity = dot / (mag1 * mag2 || 1);

```

Rysunek 15. Struktura pliku konfiguracyjnego serwer w JavaScript cz.2 - Opracowanie: Rafał Krajewski

```

JS server.js > ...
75 app.post("/check-similarity", (req, res) => {
88   phishingExamples.forEach(example => {
104
105       if (similarity > maxSimilarity) {
106         maxSimilarity = similarity;
107       }
108     });
109
110     res.json({ similarity: maxSimilarity });
111   } catch (err) {
112     console.error("❌ Błąd serwera:", err);
113     res.status(500).json({ error: "Błąd podczas przetwarzania." });
114   }
115 });
116
117
118 app.post("/check-url", (req, res) => {
119   const url = req.body.url;
120
121   try {
122     const domain = new URL(url).hostname;
123
124     const tldSuspicious = suspiciousTLDs.some(tld => domain.endsWith(tld));
125
126     const tokenizer = new natural.WordTokenizer();
127     const domainTokens = tokenizer.tokenize(domain.toLowerCase());
128
129     let maxSimilarity = 0;
130     for (const legit of knownLegitDomains) {
131       const legitTokens = tokenizer.tokenize(legit);
132       const similarity = natural.JaroWinklerDistance(domainTokens.join(" "), legitTokens.join(" "));
133       if (similarity > maxSimilarity) {
134         maxSimilarity = similarity;
135       }
136     }
137
138     const isSuspicious = tldSuspicious || maxSimilarity > 0.85;
139
140     res.json({
141       domain,
142       maxSimilarity,
143       tldSuspicious,
144       isSuspicious
145     });
146   } catch (e) {
147     console.error("❌ Błąd analizy URL:", e);
148     res.status(400).json({ error: "Nieprawidłowy adres URL" });
149   }
150 }
151 });
152
153 app.listen(port, () => {
154   console.log(`✅ Serwer działa na http://localhost:${port}`);
155 });

```

Rysunek 16. Struktura pliku konfiguracyjnego serwera w JavaScript cz.3 - Opracowanie: Rafał Krajewski

Plik zapisano w języku JavaScript. Zgodnie z rysunkami 14, 15, 16 Pełni on rolę serwera aplikacji internetowej, której zadaniem jest analiza wiadomości e-mail oraz adresów URL pod kątem wystąpienia potencjalnego phishingu. Dodatkowo zaimplementowana została funkcjonalność służąca do generowania wiadomości e-mail o charakterze phishingowym oraz wysyłania jej na podany adres e-mail. Zrealizowane zostały trzy główne funkcjonalności, czyli sprawdzenie podobieństwa treści e-mail do

znanych i zadeklarowanych przykładów phishingowych, weryfikacja domen URL pod kątem podejrzanych końcówek adresu URL i podobieństwa do wcześniej zadeklarowanej zmiennej w pliku „const.js” opisującej przykłady legalnych domen oraz wysyłanie testowego e-maila phishingowego.

W siedmiu pierwszych liniach pliku zadeklarowane są zmienne `const`, które importują wymagane biblioteki i dane. Pierwsza zmienna (`const express = require("express");`) dodaje bibliotekę `express` do budowy serwera HTTP, której zadaniem jest usprawnienie pracy i rozszerzenie funkcjonalności Node.js. Druga zmienna (`const cors = require("cors");`) dodaje mechanizm, który obsługuje udostępnianie zasobów między domenami. Trzecia zmienna (`const bodyParser = require("body-parser");`) dodaje funkcję, która analizuje oraz przekształca dane przesłane poprzez żądania HTTP. Czwarta zmienna (`const natural = require("natural");`) dodaje bibliotekę, której zadaniem jest przetwarzanie języka naturalnego. Piąta zmienna (`const nodemailer = require("nodemailer");`) dodaje bibliotekę do wysyłania e-mail przy użyciu serwera SMTP. Szósta zmienna (`const {spawn} = require("child_process");`) pozwala na uruchomienie zewnętrznych procesów systemowych, takich jak modele językowe. Ostatnia siódma zmienna importuje dane z pliku (`const { phishingExamples, suspiciousTLDs, knownLegitDomains } = require("./const");`), które zawierają przykładowe treści e-maila phishingowego, podejrzane zakończenia domen oraz listę popularnych bezpiecznych domen.

W kolejnych liniach tworzony jest obiekt serwera (`const app = express();`) oraz jest przypisywany port (`const port = 3000;`). Następnie aktywowany jest mechanizm do obsługi `cors` oraz parsowania JSON, czyli przekształcanie tekstu na obiekty lub tablice, które można następnie wykorzystać w aplikacji internetowej.

Następnie zdefiniowana została funkcja asynchroniczna „`genetareEmail(prompt)`”, która służy do tworzenia realistycznych wiadomości phishingowych. Wewnątrz tej funkcji tworzony jest nowy proces systemowy „`spawn`” z wykorzystaniem narzędzia sztucznej inteligencji Ollama oraz modelu `mistral`, który odpowiada za generowanie treści wiadomości e-mail. Do standardowego wejścia tego procesu przekazywany jest „`prompt`”, czyli tekstowa instrukcja opisująca, jak ma wyglądać wygenerowany e-mail. Następnie funkcja nasłuchuje dane wpisywane przez użytkownika i zapisuje je do zmiennej „`output`”. Po zakończeniu działania procesu,



uzyskana treść zostaje zwrócona jako wynik funkcji „resolve(output.trim())”. W przypadku wystąpienia błędu, w konsoli wyświetlany jest komunikat „Błąd generowania e-maila”, a funkcja zwraca losowy e-mail phishingowy pobrany z tablicy „fakeEmails”.

Dalej skonfigurowano obiekt „transporter” przy użyciu biblioteki „nodemailer”. Do wykonania tego użyto usługi „Gmail”, wraz z danymi do logowania. Są one wypisane jawnie w kodzie, ze względu iż takie rozwiązanie umożliwiło szybkie i bezpośrednie uruchomienie funkcjonalności bez konieczności implementowania dodatkowych mechanizmów zarządzania konfiguracją. Zastosowanie takiego podejścia miało na celu uproszczenie procesu testowania oraz przyspieszenie realizacji w warunkach środowiska deweloperskiego. Należy jednak podkreślić, że tego rodzaju praktyka nie jest zalecana ze względu na istotne ryzyko związane z bezpieczeństwem.

Następnie zdefiniowany został pierwszy endpoint (app.post(„/send-email”, [...])), który odpowiada za wygenerowanie i wysłanie testowej wiadomości phishingowej e-mail na wskazany przez użytkownika adres. W pierwszej kolejności z żądania pobierana jest zmienna „email”. Jeżeli nie zostanie ona przesłana do serwera, zwracany jest błąd „Brak adresu email!”. W przypadku prawidłowego przesłania zmiennej, wywoływana jest funkcja „generateEmail()”. Funkcja ta otrzymuje rozbudowany prompt, którego treść brzmi *„Napisz przekonujący, realistyczny e-mail phishingowy, który wygląda jakby był wysłany przez instytucję finansową lub dużą firmę. Użyj alarmującego tonu, który nakłania odbiorcę do natychmiastowego działania, np. kliknięcia w link lub podania danych. Link powinien wyglądać na zaufany (np. 'mbank.pl'), ale prowadzić gdzie indziej (np. 'login-verification-info.net'). Użyj wymyślonych, lecz naturalnie brzmiących danych — nie używaj nawiasów ani oznaczeń typu [imię]. Nie używaj słów takich jak subiekt, subject, subiekt ani podobnych. Styl i język mają być jak najbardziej zbliżone do autentycznych wiadomości phishingowych.”*. Po wygenerowaniu wiadomości tworzona jest zmienna „mailOptions”, zawierająca adres nadawcy, adres odbiorcy określony przez użytkownika, temat wiadomości „Ważna wiadomość” oraz treść e-maila. Następnie wykonywana jest próba wysłania e-maila, przy użyciu wcześniej skonfigurowanego transportera. Jeżeli uda się wysłać e-mail wyświetlany jest komunikat „E-mail został wysłany!”. Natomiast w przypadku błędu, w konsoli zapisywany jest komunikat „Błąd wysłania e-maila:” z treścią błędu, a użytkownikowi zwracany jest komunikat o niepowodzeniu „Wystąpił błąd podczas wysyłania phishingu.” wraz z kodem błędu „500”.

Drugi endpoint (`app.post(„/check-similarity”, [...])`) odpowiada za sprawdzenie, czy przesłana treść wiadomości e-mail jest podobna do znanych przykładów phishingu. W pierwszej kolejności z żądania pobierana jest treść e-mail „emailText”. Jeżeli nie zostanie przesłany do serwera, na stronie internetowej zwracany jest błąd 400, który wyświetla błąd „Brak tekstu e-maila.”. W przypadku pobrania danych, są one tokenizowane[26] przy użyciu `WordTokenizer` czyli podstawowego elementu języka JavaScript, który rozbija ciąg znaków lub słów na mniejsze jednostki, zwane tokenami. Następnie jest to przekształcane na wektor cech z wykorzystaniem modelu „tfidf”[27]. Dla każdego z zadeklarowanych przykładów phishingu obliczany jest wyżej wymieniony wektor „tfidf”, a następnie miara podobieństwa kosinusowego. Po tych obliczeniach zwracana jest największa uzyskana wartość podobieństwa „maxSimilarity” w odpowiedzi jako wynik. Jeżeli wystąpi jakiś error, w konsoli wyświetla się komunikat „Błąd serwera:”, a na stronie internetowej pojawia się informacja „Błąd podczas przetwarzania.”

Trzeci endpoint (`app.post(„/check-url”, [...])`) odpowiada za analizę adresu URL pod kątem phishingu. W pierwszej kolejności z wpisanego przez użytkownika adresu URL wyodrębniana jest nazwa domeny. Kolejnym krokiem jest sprawdzenie, czy końcówka domeny znajduje się na liście podejrzanych domen. Następnie domena jest tokenizowana i porównywana ze ztokenizowanymi domenami, które zostały wcześniej zadeklarowane. Kolejnym krokiem jest zamiana tokenów domeny na małe litery alfabetu. Do obliczenia miary podobieństwa między domenami jest wykorzystywana odległość Jaro-Winklera. Jeżeli podejrzana końcówka adresu URL „tldSuspicious” występuje lub podobieństwo przekracza próg 0.85, domena uznawana jest za potencjalnie podejrzaną „isSuspicious”.

Ostatnia część pliku służy do uruchomienia i nasłuchiwania serwera, czyli reagowania na działania użytkownika, takie jak kliknięcia myszą, naciśnięcie klawiszy klawiatury czy przewijania strony.



```
PS C:\Users\rkrajewski\OneDrive - opi.org.pl\Pulpit\Nowy folder (2)\PRACA INŻYNIERSKA\Kody> node .\server.js
✓ Server działa na http://localhost:3000
```

Rysunek 17. Włączenie serwera Node.js - Opracowanie: Rafał Krajewski

Zgodnie z rysunkiem 17 aby uruchomić „server.js”, trzeba włączyć aplikację systemową „Windows PowerShell”, a następnie przejść do folderu, w którym znajduje się wyżej wymieniony plik. Włączamy serwer wpisując „node .\server.js”. Po uruchomieniu na ekranie wyświetla się komunikat „Serwer działa na http://localhost:3000”.

### 2.1.13 Plik skryptu.js

```
JS skrypty.js > document.addEventListener("DOMContentLoaded") callback > then() callback
1 console.log("✅ Skrypt `skrypty.js` został załadowany!");
2
3 document.addEventListener("DOMContentLoaded", () => {
4     const secpage = document.getElementById("section-placeholder");
5
6     if (secpage) {
7
8         fetch("sec.html")
9             .then(response => response.text())
10            .then(data => {
11                secpage.innerHTML = data;
12                console.log("✅ Plik `sec.html` został załadowany!");
13
14                const phishingMailForm = document.getElementById("phishing-mail");
15                if (!phishingMailForm) {
16                    console.error("❌ Nie znaleziono formularza o ID 'phishing-mail' po załadowaniu sec.html");
17                    return;
18                }
19
20                console.log("✅ Formularz został znaleziony:");
21
22                phishingMailForm.addEventListener("submit", async (e) => {
23
24                    e.preventDefault();
25                    console.log("✅ Submit event zadziałał!");
26                    const email = document.getElementById("email").value;
27
28                    try {
29                        const response = await fetch("http://localhost:3000/send-email", {
30                            method: "POST",
31                            headers: { "Content-Type": "application/json" },
32                            body: JSON.stringify({ email }),
33                        });
34
35                        const result = await response.json();
36                        alert(result.message);
37                    } catch (error) {
38                        console.error("Błąd:", error);
39                        alert("❌ Wystąpił błąd podczas wysyłania phishingu.");
40                    }
41                });
42            })
43            .catch(error => console.error("❌ Błąd ładowania sekcji w main:", error));
44        } else {
45            console.error("Nie znaleziono elementu o ID 'section-placeholder'");
46        }
47    });
48
49    console.log("✅ Skrypt `phishing-mail` został załadowany!");
50
51    const navpage = document.getElementById("nav-placeholder");
52    if (navpage) {
53        fetch("nav.html")
54            .then(response => response.text())
55            .then(data => {
```

Rysunek 18. Struktura pliku skryptowego w JavaScript cz.1 - Opracowanie: Rafał Krajewski



Plik zapisano w języku JavaScript. Pełni on kluczową rolę w funkcjonowaniu interfejsu strony internetowej poprzez dynamiczne zarządzanie treścią oraz obsługę interakcji użytkownika. Zgodnie z rysunkami 18 i 19 podstawowym zadaniem tego pliku jest dynamiczne doładowywanie zewnętrznych plików napisanych w języku HTML do określonych sekcji strony. W pierwszej linijce skryptu, za pomocą funkcji „console.log()”, wypisywana w konsoli przeglądarki jest informacja, że „Skrypt ‘skrypty.js’ został załadowany!”. Kolejnym krokiem jest załadowanie struktury DOM, czyli obiektowego modelu dokumentu, a to daje nam możliwość tworzenia dynamicznych stron internetowych[28].

W pierwszej części skryptu znajduje się dynamiczne ładowanie sekcji HTML „sec.html”. Za pomocą „document.getElementById(„section-placeholder”)” pobierany jest element, do którego zostanie wczytana zawartość pliku „sec.html”. Jeżeli wyżej wymieniony plik istnieje, wykonane zostanie żądanie „fetch(„sec.html”)”, czyli pobranie danych. Następnie w konsoli przeglądarki zostaje wpisana treść „Plik ‘sec.html’ został załadowany!”. Gdy sekcja „sec.html” zostanie załadowana pobierany jest formularz z identyfikatorem „phishing-mail”. Następnie wykonywane jest sprawdzenie czy udało się odnaleźć wcześniej wspomniany formularz. Jeżeli nie, to w konsoli przeglądarki zostaje zapisany error o treści „Nie znaleziono formularza o ID 'phishing-mail' po załadowaniu sec.html”. Natomiast jak uda się wyszukać formularz, to w konsoli przeglądarki zostaje zapisana treść „Formularz został znaleziony”. Jeżeli wspomniany formularz zostanie odnaleziony, przypisywany jest do niego nasłuchiwaniec ([...]addEventListener[...]), który ma za zadanie wyłapać czy użytkownik witryny internetowej nacisnął przycisk „submit”. Następnie w konsoli przeglądarki zostaje zapisana treść „Submit event zadziałał”. Przy wysłaniu formularza wyłączana jest domyślna akcja „preventDefault”. Z pola o identyfikatorze email, pobierana jest wartość, która następnie jest zapisywana w zmiennej email. Następnie wysyłane jest żądanie POST do http://localhost:3000/send-email z nagłówkiem „Content-Type: application/json” i pobranym adresem e-mail. Gdy otrzymamy odpowiedź wyświetlany jest komunikat na stronie z otrzymaną odpowiedzią. W przypadku błędu, w konsoli przeglądarki zapisywana jest treść erroru i na stronie internetowej wyświetlany jest alert „Wystąpił błąd podczas wysyłania phishingu”. Natomiast jeżeli nie uda się załadować całej sekcji „sec.html”, zostaje zapisany w konsoli error o treści „Błąd ładowania sekcji w main:”. Dodatkowo gdy nie uda się załadować elementu o identyfikatorze „section-placeholder” zostaje w konsoli zapisany error o treści

„Nie znaleziono elementu o ID ‘section-placeholder’”. Jeżeli wszystko zadziała w konsoli przeglądarki zostaje zapisana treść „Skrypt ‘phishing-mail’ został załadowany!”

W drugiej części skryptu znajduje się dynamiczne ładowanie sekcji HTML „nav.html”. Za pomocą „document.getElementById(„nav-placeholder”)” pobierany jest element, do którego zostanie wczytana zawartość pliku „nav.html”. Następnie sprawdzany jest element „nav-placeholder”, jeśli istnieje wykonane będzie żądanie „fetch(„nav.html”)”. Jeżeli wystąpi błąd, w konsoli przeglądarki zostaje zapisany error o treści „Błąd ładowania nawigacji”, natomiast jak zadziała wszystko, w konsoli witryny zostaje zapisana treść „Plik ‘nav.html’ został załadowany”.

W trzeciej części skryptu znajduje się dynamiczne ładowanie sekcji HTML „carousel.html”. Za pomocą „document.getElementById(„carousel-placeholder”)” pobierany jest element, do którego zostanie wczytana zawartość pliku „carousel.html”, jeśli wyżej wymieniony element nie istnieje, w konsoli przeglądarki zostanie zapisany error o treści „Nie znaleziono elementu #carousel-placeholder! Sprawdź HTML.”. Natomiast jeżeli ten element istnieje będzie wykonane żądanie „fetch(„carousel.html”)” i w konsoli przeglądarki zostanie zapisana treść „Element #carousel-placeholder” znaleziony!”. Jednakże jeżeli nie uda się wykonać tego żądania, zostaje zapisany błąd o treści „Błąd http:” i po dwukropku informacja o błędzie. Następnie zawartość pliku „carousel.html” jest przypisywana do elementu „carousel-placeholder”, i w konsoli przeglądarki zostaje zapisana treść „Karuzela załadowana!”. Po 100ms uruchamiana jest funkcja „initializeCarousel()”. Jeżeli nie uda się załadować zawartości pliku, w konsoli zostaje zapisany error o treści „Błąd ładowania karuzeli:” i po dwukropku informacja o błędzie.

W ostatniej części skryptu znajduje się funkcja, której zadaniem jest cyklicznie przewijać elementy karuzeli. Na początku funkcji pobierane są elementy „carousel” oraz „carousel-item”. Jeżeli nie uda się znaleźć elementu wyżej wymienionych elementów, w konsoli zostaje zapisany error o treści „Brak klasy „carousel” lub elementów „carousel-item”!”. Następnie deklarowana jest zmienna „currentIndex”, której zadaniem jest ustawić początkowy index o wartości 0. Kolejno jest definiowana funkcja „showNextItem()”, której zadaniem jest pokazanie na stronie internetowej następny element karuzeli. Jeżeli element „currentIndex” nie istnieje, w konsoli zostaje zapisany error o treści „Element z indeksem \${currentIndex} nie istnieje!”, gdzie „\${currentIndex}” oznacza aktualnie załadowany element karuzeli. Następnie ze zmiennej jest usuwana klasa „active” i wykonuje się przejście do następnego elementu

karuzeli. Jeżeli element „currentIndex” nie istnieje, w konsoli zostaje zapisany error o treści „Element z indeksem \${currentIndex} nie istnieje!”, a następnie do zmiennej dodawana jest klasa „active”. Po tym jest ustawiany interwał czasowy, który ma za zadanie wykonać tą funkcję co 3 sekundy. Jeżeli wszystko zadziała, w konsoli zostaje zapisana treść „Plik „carousel.html” został załadowany!”.

### 2.1.14 Plik sprawdzeniephi.html

```
<> sprawdzeniephi.html > html > body > main > section#phi > h2
1  <!DOCTYPE html>
2  <html lang="pl">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Praca Inżynierska</title>
7      <link rel="stylesheet" href="style.css">
8      <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css">
9      <script src="skrypty.js" defer></script>
10 </head>
11 <body>
12     <header>
13         <h1>Projekt narzędzia do analizy phishingu i implementacja w technologii JavaScript</h1>
14         <div id="nav-placeholder"></div>
15     </header>
16     <main>
17         <section id="phi">
18             <h2>Wykrywanie potencjalnego phishingu, wklej treść e-maila poniżej:</h2>
19             <form id="phishing-form">
20                 <textarea id="email-content" rows="8" cols="60" placeholder="Wklej e-mail tutaj"></textarea><br>
21                 <button type="submit">Sprawdź</button>
22             </form>
23             <div id="result" style="margin-top:20px;"></div>
24
25             <script src="phishingrozpoznanie.js"></script>
26         </section>
27
28
29
30
31
32     </main>
33     <footer>
34         <p>&copy; 2025 Praca Inżynierska - Rafał Krajewski</p>
35     </footer>
36 </body>
37 </html>
```

Rysunek 20. Struktura pliku do analizy treści e-mail w HTML - Opracowanie : Rafał Krajewski

Plik zapisano w języku HTML. Zgodnie z rysunkiem 20 zawiera on strukturę strony internetowej, która jest identyczna jak pliku „ochronaphi.html”, „phi.html”, „rozpoznaniephi.html”, jednakże różni się ona tym co się znajduje w znaczniku (<main>). Witryna ta ma na celu analizę treści e-mail, która zostanie wpisana przez użytkownika.

Główna część strony została umieszczona w znaczniku (<main>), a w sekcji (<section id="phi">) znajduje się podtytuł „Wykrywanie potencjalnego phishingu, wklej treść e-maila poniżej. Poniżej znajduje się formularz o identyfikatorze „phishing-form”, w którym zlokalizowane jest pole tekstowe o identyfikatorze „email-content” i podpowiedź o treści „Wklej e-mail tutaj”. Umożliwia się wpisanie przez użytkownika podejrzaną treść e-mail do wyżej wymienionego pola tekstowego do analizy. Poniżej znajduje się przycisk „submit”, na którym jest napisana treść „Sprawdź”, który ma za zadanie wysłać do serwera wpisaną przez nas treść e-mail, aby została sprawdzona. Pod formularzem znajduje się znacznik (<div>), który ma za zadanie określić miejsce gdzie będzie wyświetlony rezultat analizy phishingu. W ostatniej linijce sekcji (<main>) znajduje się podłączenie do skryptu (<script src="phishingrozpoznanie.js">).

### 2.1.15 Plik sprawdzenieurl.html

```
<> sprawdzenieurl.html > html > body > main > section#phi > h2
1  <!DOCTYPE html>
2  <html lang="pl">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Praca Inżynierska</title>
7      <link rel="stylesheet" href="style.css">
8      <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css">
9      <script src="skrypty.js" defer></script>
10 </head>
11 <body>
12     <header>
13         <h1>Projekt narzędzia do analizy phishingu i implementacja w technologii JavaScript</h1>
14         <div id="nav-placeholder"></div>
15     </header>
16     <main>
17         <section id="phi">
18             <h2>Wykrywanie podejrzanego adresu URL, wklej go poniżej:</h2>
19             <form id="phishing-form">
20                 <input type="text" id="url-content" placeholder="Wklej podejrzaną link" /> <br><br>
21                 <button type="submit">Sprawdź</button>
22             </form>
23
24             <div id="result" style="margin-top:20px;"></div>
25
26             <script src="urlrozpoznanie.js"></script>
27         </section>
28
29
30
31
32     </main>
33     <footer>
34         <p>&copy; 2025 Praca Inżynierska - Rafał Krajewski</p>
35     </footer>
36 </body>
37 </html>
```

Rysunek 21. Struktura pliku do analizy adresu URL w HTML - Opracowanie : Rafał Krajewski



Plik zapisano w języku HTML. Zgodnie z rysunkiem 21 zawiera on strukturę strony internetowej, która jest identyczna jak pliku „ochronaphi.html”, „phi.html”, „rozpoznaniephi.html”, „sprawdzeniephi.html” jednakże różni się ona tym co się znajduje w znaczniku (<main>). Witryna ta ma na celu analizę adresu URL, który zostanie wpisany przez użytkownika.

Główna część strony została umieszczona w znaczniku (<main>), a w sekcji (<section id=„phi”>) znajduje się podtytuł „Wykrywanie podejrzanego adresu URL, wklej go poniżej:”. Poniżej znajduje się formularz o identyfikatorze „phishing-form”, w którym zlokalizowane jest pole tekstowe o identyfikatorze „url-content” i odpowiedź o treści „Wklej podejrzany link”. Umożliwia się wpisanie przez użytkownika podejrzany adres URL do wyżej wymienionego pola tekstowego do analizy. Poniżej znajduje się przycisk „submit”, na którym jest napisana treść „Sprawdź”, który ma za zadanie wysłać do serwera wpisany przez nas adres, aby został sprawdzony. Pod formularzem znajduje się znacznik (<div>), który ma za zadanie określić miejsce gdzie będzie wyświetlony rezultat analizy phishingu. W ostatniej linijce sekcji (<main>) znajduje się podłączenie do skryptu (<script src=„urlrozpoznanie.js”>).

### 2.1.16 Plik strona.html

```
<> strona.html > ...
1  <!DOCTYPE html>
2  <html lang="pl">
3  <head>
4      <meta charset="UTF-8">
5      <meta name="viewport" content="width=device-width, initial-scale=1.0">
6      <title>Praca Inżynierska</title>
7      <link rel="stylesheet" href="style.css">
8      <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.4.2/css/all.min.css">
9      <script src="skrypty.js" defer></script>
10 </head>
11 <body>
12     <header>
13         <h1>Projekt narzędzia do analizy phishingu i implementacja w technologii JavaScript</h1>
14         <div id="nav-placeholder"></div>
15     </header>
16     <div id="carousel-placeholder"></div>
17     <main>
18         <div id="section-placeholder"></div>
19     </main>
20     <footer>
21         <p>&copy; 2025 Praca Inżynierska - Rafał Krajewski</p>
22     </footer>
23
24 </body>
25 </html>
```

Rysunek 22. Struktura pliku witryny internetowej w HTML - Opracowanie: Rafał Krajewski

Plik zapisano w języku HTML. Zgodnie z rysunkiem 22 zawiera on strukturę strony internetowej, która jest identyczna jak pliku „ochronaphi.html”, „phi.html”, „rozpoznaniephi.html”, „sprawdzeniephi.html”, „sprawdzenieurl.html” jednakże różni się ona tym co się znajduje w znaczniku (<body>). Witryna ta pełni rolę strony głównej całej aplikacji.

Główna część strony znajduje się w znaczniku (<body>). W środku znajdują się elementy takie jak:

- (<div id=„nav-placeholder”>)
- (<div id=„carousel-placeholder”>)
- (<div id=„section-placeholder”>)

Pierwszy element pełni rolę kontenera dla dynamicznie ładowanego menu nawigacyjnego. Drugi element służy jako miejsce, w którym dynamicznie zostanie wczytana karuzela z obrazkami. Ostatni z wyżej wymienionych elementów pełni funkcję kontenera dla dynamicznego ładowania zawartości merytorycznej.

## 2.1.17 Plik style.css

```
# style.css > #phi
1 body {
2     display: flex;
3     flex-direction: column;
4     height: 100vh;
5     margin: 0;
6     padding: 0;
7     font-family: 'Trebuchet MS';
8     background-color: #blanc;
9 }
10
11 main {
12     flex: 1;
13     display: flex;
14     justify-content: space-between;
15     align-items: center;
16     flex-direction: column;
17     margin: auto;
18     padding-top: 45px;
19     padding-bottom: 160px;
20 }
21
22 #section-placeholder {
23     display: flex;
24     flex-wrap: wrap;
25     justify-content: wrap;
26     margin-left: 20%;
27     margin-right: 20%;
28 }
29
30 #phi, #rozpoznanie {
31     justify-content: wrap;
32     margin-left: 20%;
33     margin-right: 20%;
34 }
35
36 @media (max-width: 768px) {
37     body {
38         font-size: 16px;
39     }
40
41     nav ul {
42         flex-direction: column;
43         align-items: flex-start;
44         gap: 5px;
45     }
46
47     nav ul li {
48         display: block;
49         width: 100%;
50     }
51
52     h1 {
53         font-size: 20px;
54         text-align: center;
55 }
56
57 #style.css > #phi
58 @media (max-width: 768px) {
59     .carousel {
60         width: 95%;
61     }
62
63     #section-placeholder,
64     #phi,
65     #rozpoznanie {
66         flex-direction: column;
67         margin: 0 auto;
68         padding: 0 10px;
69     }
70
71     input, button, textarea {
72         width: 100%;
73         box-sizing: border-box;
74         margin-bottom: 10px;
75     }
76
77     form {
78         width: 100%;
79     }
80
81     footer {
82         position: fixed;
83         bottom: 0;
84         left: 0;
85         width: 100%;
86         text-align: center;
87         padding: 10px 0;
88         background-color: #f1f1f1;
89     }
90
91     h1, nav {
92         background-color: #burlywood;
93         padding: 10px;
94         margin: 0;
95         display: block;
96         width: 100%;
97     }
98
99     nav ul {
100         list-style: none;
101         padding: 0;
102         margin: 0;
103         display: flex;
104         gap: 15px;
105     }
106
107     nav ul li {
108         display: inline;
109 }
```

Rysunek 23. Struktura pliku reguł formatowania w CSS cz.1 - Opracowanie: Rafał Krajewski

```

# style.css > .home-button
110 nav ul li a {
111     text-decoration: none;
112     color: #36454f;
113     font-weight: bold;
114     padding: 5px 10px;
115     transition: color 0.3s ease-in-out;
116 }
117
118 li {
119     padding: 5px;
120 }
121
122 a:hover {
123     color: darkred;
124 }
125
126 .home-button {
127     display: inline-flex;
128     align-items: center;
129     justify-content: center;
130     font-size: 17px;
131     color: black;
132     text-decoration: none;
133     border: none;
134     background: none;
135     cursor: pointer;
136     transition: color 0.3s ease-in-out;
137 }
138
139 .home-button:hover {
140     color: darkred;
141 }
142
143 .carousel {
144     position: relative;
145     width: 80%;
146     max-width: 600px;
147     margin: auto;
148     overflow: hidden;
149     margin-top: 120px;
150 }
151
152 .carousel-item {
153     display: none;
154     width: 100%;
155     text-align: center;
156 }
157
158 .carousel-item.active {
159     display: block;
160 }
161
# style.css > .home-button
162 .carousel img {
163     height: auto;
164     border-radius: 10px;
165 }
166
167
168 .carousel-caption {
169     position: absolute;
170     bottom: 10px;
171     left: 50%;
172     transform: translateX(-50%);
173     background: rgba(0, 0, 0, 0.5);
174     color: white;
175     padding: 5px 10px;
176     border-radius: 5px;
177 }
178

```

Rysunek 24. Struktura pliku reguł formatowania w CSS cz.2- Opracowanie: Rafał Krajewski

Plik zapisano w języku CSS. Zsawiera on strukturę która pełni rolę definiowania stylów wyglądu strony internetowej. Stylizacja została podzielona na kilkadziesiąt sekcji, z których każda z nich odpowiada za inny komponent strony. Powyższy plik jest jednym z najważniejszych plików do poprawnego wyświetlania się witryny internetowej.

Na początku zgodnie z rysunkiem 23 ustawiono styl dla znacznika (`<body>`), w którym zastosowano układ oparty na fleksboksie (`display: flex`), co umożliwia wygodne rozmieszczanie jego „dzieci” (np.: `main`, `footer`) w pionie lub poziomie. Dla elementów potomnych ustawiono kierunek wyświetlania danych jako kolumna, czyli jeden pod drugim (`flex-direction: column`). Cała wysokość widoku witryny została przypisana jako wysokość elementu (`height: 100vh`) co oznacza 100% wysokości widocznego obszaru okna przeglądarki. Usunięto marginesy i wewnętrzne odstępy (`margin: 0; padding: 0`), aby zminimalizować niechciane przesunięcia. Zdefiniowana została również czcionka (`font-family: 'Trebuchet MS', sans-serif`), a w przypadku braku dostępności tej czcionki używana jest czcionka szeryfowa (np.: `Times New Roman`). Na całej powierzchni znacznika (`<body>`) został ustawiony kolor tła (`background-color: blanchedalmond`).

W kolejnym elemencie ustawiono styl dla znacznika (`<main>`), który również korzysta z fleksboksowego układu, jednakże wartość (`flex: 1`) oznacza zajmowanie przez tą sekcję całej dostępnej przestrzeni jaka pozostała. Zdefiniowane zostało wyrównanie elementów w poziomie (`align-items: center`) i pionie (`justify-content: space-between`). Dodano ustawienia marginesu automatycznie centrowanego w poziomie (`margin: auto`), odstępy wewnętrzne od góry (`padding-top: 45px`) i dołu (`padding-bottom: 160px`).

W kolejnej sekcji ustawiono styl dla identyfikatora (`<#section-placeholder>`), w którym zdefiniowano elastyczne rozmieszczenie elementów (`display: flex`) oraz w razie potrzeby elementy będą zawijane do następnego wiersza (`flex-wrap: wrap`). Dodatkowo zastosowany został styl, który wyrównuje elementy (`justify-content: wrap`). Ustawione zostały również marginesy boczne (`margin-left: 20%; margin-right: 20%`) na wartość 20% szerokości elementu, co oznacza odsunięcie od lewej i prawej krawędzi strony internetowej o jedną piątą szerokości całej sekcji.

Dla identyfikatorów (`<#phi, #rozpoznanie>`) ustawiony został styl taki sam jak dla powyższego akapitu, jednakże bez definiowania elastycznego rozmieszczenia elementów na stronie internetowej.

W kolejnej sekcji ustawiono styl dla urządzeń mobilnych (@media) dla ekranów o maksymalnej szerokości 768 pikseli(px). Zastosowano zmniejszenie rozmiaru czcionki w sekcji (<body>) do 16px w celu poprawienia czytelności i dostosowania do mniejszych wyświetlaczy. Kolejna część sekcji dla urządzeń mobilnych (<nav ul>) odnosi się do nawigacji i listy nienumerowanej. Ustawia ona układ elementów listy (ul) w kolumnie jeden pod drugim oraz ustawia wyrównanie elementów do lewej krawędzi kontenera. Dodatkowo ustawiony został odstęp o wartości 5px między sąsiadującymi elementami listy. Kolejną częścią sekcji dla urządzeń mobilnych jest stylizacja (<nav ul li>), która sprawia, że każdy element listy (li) zachowuje się jak blok, zajmując całą dostępną szerokość. W dodatku ustawiona została stała szerokość każdego elementu listy na 100% szerokości jego kontenera (ul). W kolejnej części sekcji dla urządzeń mobilnych zastosowano stylizację dla nagłówka (<h1>), która ustawia rozmiar czcionki nagłówka na 20px oraz wyśrodkowuje treść nagłówka w poziomie. Kolejną częścią sekcji dla urządzeń mobilnych jest stylizacja klasy (<.carousel>), która ustanawia szerokość elementu na 95% szerokości elementu, w którym się znajduje. Następną częścią sekcji dla urządzeń mobilnych jest stylizacja (<#section-placeholder, #phi, #rozpoznanie>), która ustawia kierunek układu potomnych elementów wewnątrz tych kontenerów na kolumnowy. Dodatkowo ustawia margines górny i dolny na 0, a boczne na auto, co powoduje poziome wyśrodkowanie tych sekcji oraz dodaje poziomy wewnętrzny odstęp po 10px z każdej strony, bez wpływu na górę i dół. W kolejnej części sekcji dla urządzeń mobilnych znajduje się stylizacja pól formularza i przycisków (<input, button, textarea>), która ustanawia 100% dostępnej szerokości kontenera oraz zmienia sposób obliczania elementu w taki sposób, że w skład szerokości wlicza się padding i border. Dodatkowo dodawany jest dolny margines o wartości 10px, co powoduje odstęp między kolejnymi elementami. Ostatnią częścią sekcji dla urządzeń mobilnych jest stylizacja formularzy (<form>), która ustawia wartość swojego kontenera na 100% dostępnej szerokości.

W kolejnej sekcji ustawiono styl dla stopki (<footer>), który powoduje „przyklejenie” jej do dolnej krawędzi okna przeglądarki niezależnie od przewijania. Dodatkowo ustawiono dokładne położenie elementu w lewym dolnym rogu oraz szerokość, która zajmuje całą szerokość okna witryny internetowej. Zastosowano również wyśrodkowanie całej zawartości stopki, z uwzględnieniem wewnętrznego odstępu na 10px bez poziomego oraz ustawiono kolor tła stopki na jasnoszary.

Kolejna sekcja ustawia styl dla nagłówka oraz nawigacji (<h1, nav>), który ustawia kolor tła na jasnobrązowy. Dodatkowo zastosowano wewnętrzne odstępy wokół zawartości z każdej strony o wartości 10px, brak zewnętrznych marginesów, zachowanie elementów jako blok oraz ustanowiona została szerokość na 100% dostępnej w obrębie kontenera.

W kolejnej sekcji ustawiono styl dla listy nienumerowanej w nawigacji (<nav ul>), który usuwa domyślne punktory listy, czyli kropki oraz usuwa domyślne wewnętrzne odstępy i marginesy. Dodatkowo zastosowano fleksboks i odstęp między elementami listy, którego wartość to 15px.

Kolejna sekcja ustawia styl dla elementów listy w nawigacji (<nav ul li>), który umieszcza każdy element z listy jako liniowy, co oznacza że będą one wyświetlane obok siebie pozwalając na stworzenie poziomego menu.

W kolejnej sekcji zgodnie z rysunkiem 24 ustawiono styl dla linków w liście (<nav ul li a>), który usuwa domyślne podkreślenie hiperłączy znajdujących się w elementach listy. W dodatku został ustawiony kolor tekstu linków na czarny.

Kolejna sekcja ustawia styl ogólny dla elementów listy (<li>), który daje każdemu elementowi z listy wewnętrzny odstęp 5px z każdej ze stron.

W kolejnej sekcji ustawiono styl dla hiperłączy (<a:hover>), który zmienia kolor linków na ciemnoczerwony po najechaniu kursorem na niego.

Kolejna sekcja ustawia styl przycisku przekierowującego do strony głównej (<.home-button>), który daje elementowi elastyczność i wyświetlanie w linii, co umożliwia centrowanie zawartości. Dodatkowo zastosowane zostało pionowe i poziome wyrównanie zawartości oraz ustawiono wielkość czcionki na 17px. Zmieniony został kolor przycisku na czarny oraz zostało usunięte podkreślenie. W dodatku usunięto również domyślną ramkę oraz tło przycisku. Zastosowano zmianę wyglądu kursora na „rękę” gdy najedziemy na guzik oraz zdefiniowano płynne przejście koloru przy interakcji.

W kolejnej sekcji ustawiono styl dla wyżej wymienionego przycisku, który zmienia kolor na ciemnoczerwony po najechaniu kursorem na niego.

Kolejna sekcja ustawia styl dla kontenera karuzeli (<.carousel>), który przypisuje pozycjonowanie względne, co umożliwia sterowanie pozycją w cztery różne strony.

Ustalona została szerokość kontenera na 80% oraz ograniczono maksymalną szerokość do 600px. Zastosowano również automatyczne centrowanie poziome oraz dodanie odstępu o wartości 120px nad karuzelą, co oznacza oddzielenie jej od górnych elementów strony. Dodatkowo zastosowano ukrycie wszelkich elementów, które wykraczają poza obszar kontenera.

W kolejnej sekcji ustawiono styl dla slajdów karuzeli (`<.carousel-item>`), który domyślnie ukrywa każdy slajd. Dodatkowo zastosowano ustawienie szerokości slajdu na 100%, co oznacza zajmowanie pełnej szerokości kontenera oraz zawartość slajdu jest wyśrodkowania w poziomie.

Kolejna sekcja ustawia styl dla aktywnego slajdu karuzeli (`<.carousel-item.active>`), który wyświetla tylko slajd z przypisaną klasą „active” i pojawia się w miejscu, gdzie pozostałe slajdy są ukryte. Dzięki temu karuzela pokazuje tylko jeden slajd na raz.

W kolejnej sekcji ustawiono styl dla obrazów w karuzeli (`<.carousel img>`), który rozciąga obraz na całą szerokość kontenera slajdu oraz jego wysokość dopasowywana jest proporcjonalnie do szerokości, zachowując oryginalne proporcje obrazu. Dodatkowo dodano zaokrąglenie rogów obrazu o promieniu 10px.

Kolejna sekcja ustawia styl podpisów do slajdu karuzeli (`<.carousel-caption>`), który powoduje że podpis jest pozycjonowany względem poprzedzającego elementu, czyli (`<.carousel>`). Dodatkowo umieszcza go 10px nad dolną krawędzią obrazu, oraz przesuwając podpis na środek szerokości kontenera i również cofa go o połowę własnej szerokości, co powoduje idealne wyśrodkowanie poziome tekstu. Zastosowano również półprzezroczyste czarne tło o przezroczystości 50%, aby poprawić kontrast tekstu. Ustawiono kolor tekstu na biały oraz dodano pionowe i poziome wypełnienie wewnętrzne, powodujące odstęp między tekstem a ramką. Zastosowano również zaokrąglenie rogów podpisu o promieniu 5px

.



## 2.1.18 Plik urlrozpoznanie.js

```
JS urlrozpoznanie.js > checkUrl
1  const form = document.getElementById("phishing-form");
2  const resultDiv = document.getElementById("result");
3
4  form.addEventListener("submit", async function (e) {
5    e.preventDefault();
6
7    const urlInput = document.getElementById("url-content").value;
8
9    resultDiv.innerHTML = "🔍 Analizuję treść...<br>";
10
11
12    const urlResult = await checkUrl(urlInput);
13
14    // URL similarity
15    if (urlResult.error) {
16      resultDiv.innerHTML += `<span style='color:orange;'>❌ Błąd analizy adresu URL.</span><br>`;
17    } else {
18      const urlSimilarityPercent = (urlResult.maxSimilarity * 100).toFixed(2);
19      if (urlResult.isSuspicious) {
20        resultDiv.innerHTML += `⚠️ <span style="color:red;">Adres URL wygląda podejrzanie: ${urlResult.domain}</span><br>`;
21      } else {
22        resultDiv.innerHTML += `✅ <span style="color:green;">Adres URL wygląda bezpiecznie: ${urlResult.domain}</span><br>`;
23      }
24      resultDiv.innerHTML += `📊 Podobieństwo domeny URL do zaufanych: ${urlSimilarityPercent}%<br>`;
25    }
26  });
27
28  async function checkUrl(urlText) {
29    try {
30      const response = await fetch("http://localhost:3000/check-url", {
31        method: "POST",
32        headers: { "Content-Type": "application/json" },
33        body: JSON.stringify({ url: urlText })
34      });
35      return await response.json();
36    } catch (error) {
37      console.error("Błąd sprawdzania URL:", error);
38      return { error: true };
39    }
40  }
```

Rysunek 25. . Struktura pliku do analizy adresu URL w JavaScript - Opracowanie : Rafał Krajewski

Plik zapisano języku JavaScript. Zgodnie z rysunkiem 25 służy on do analizy adresów URL przesyłanych przez użytkownika strony internetowej w formularzu. Celem tego pliku jest ocena podobieństwa domeny względem wcześniej zadeklarowanych zaufanych domen oraz wykrycie potencjalnie podejrzanych adresów.

W pierwszych dwóch liniach skryptu, są zdefiniowane dwie zmienne „const”, które służą do pobierania elementów HTML ze strony internetowej na podstawie ich identyfikatorów „id”.

Pierwsza zmienna (const form = document.getElementById(„phishing-form”)) odwołuje się do formularza (<form>) o identyfikatorze „phishing-form”. Jest to formularz, w którym umożliwia się wpisanie przez użytkownika adres URL do analizy. Zmienna „form” służy następnie do przypisania obsługi zdarzenia (w tym przypadku „submit”), które wywołuje funkcję sprawdzającą adres URL pod kątem potencjalnego zagrożenia.

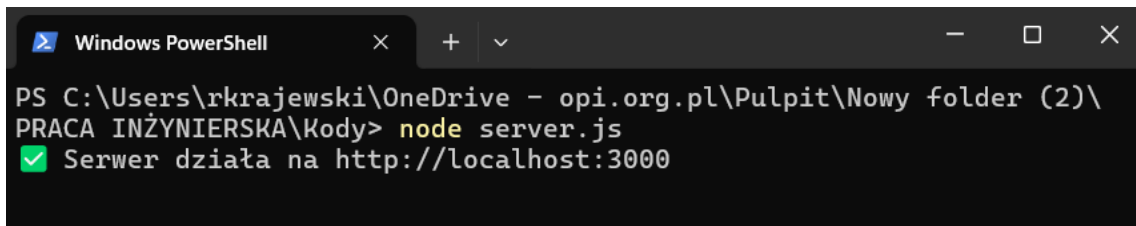
Druga zmienna (`const resultDiv = document.getElementById(„result”)`) wskazuje na element (`<div>`), którego identyfikator to „result”. Jest to miejsce, w którym będą wyświetlane wyniki analizy.

Dalsza część kodu odpowiada za przypisanie zdarzenia „submit” do formularza. W momencie kliknięcia przez użytkownika przycisku, zostaje wywołana funkcja asynchroniczna. Kluczowym elementem tej funkcji jest „`e.preventDefault()`”, która uniemożliwia domyślne odświeżanie strony, co pozwala zachować płynność działania interfejsu i umożliwia wykonanie dalszej logiki kodu bez przerw. Następnie zostają pobrane dane wpisane przez użytkownika w polu „url-content” poprzez zmienną „`const urlInput = document.getElementById(„url-content”).value`”. Kolejno w elemencie „resultDiv” tymczasowo wyświetlany jest komunikat „Analizuje treść...” co informuje użytkownika o rozpoczęciu procesu analizy. Następnie wywoływana jest funkcja „checkUrl”, która jako argument przyjmuje wartość wpisanego adresu URL i zwraca wynik analizy. Jeśli wystąpi błąd w komunikacji z serwerem, to w polu „resultDiv” zostanie wyświetlony komunikat „Błąd analizy adresu URL.”. Jeżeli z serwera zostanie przesłana poprawna odpowiedź, wykonywana jest dalsza analiza wyników. W pierwszej kolejności obliczany jest procentowy wskaźnik podobieństwa adresu URL do listy zaufanych domen. Wartość ta jest zaokrąglana do dwóch miejsc po przecinku i wyrażana w procentach. Na podstawie danych zwróconych przez serwer wykonywana jest decyzja, czy analizowany adres URL wygląda na podejrzany. Jeżeli odpowiedź wskazuje na potencjalne zagrożenie, zostanie wyświetlony komunikat „Adres URL wygląda podejrzanie.” oraz nazwa domeny. Natomiast gdy nie zostanie zakwalifikowany jako podejrzany, użytkownik otrzymuje komunikat o bezpieczeństwie adresu: „Adres URL wygląda bezpiecznie.” oraz wypisana zostaje nazwa domeny. Niezależnie od statusu bezpieczeństwa, zostaje wyświetlona informacja procentowa o dokładnym poziomie podobieństwa adresu URL do zaufanych domen.

W ostatniej sekcji kodu wykonywana jest funkcja asynchroniczna „`checkUrl(urlText)`”, która odpowiada za komunikację z lokalnym serwerem. Funkcja ta wykorzystuje metodę „`fetch`”, wykonując żądanie POST do adresu „`http://localhost:3000/check-url`”. Przesyłana treść zawiera adres URL wpisany przez użytkownika przekonwertowany na format JSON. Odpowiedź serwera przekształcona jest do formatu JSON i zwrócona jako wynik funkcji „`return await response.json()`”. Jeżeli wystąpi błąd podczas wykonywania żądania, funkcja obsługuje wyjątek „`catch`”

zapisując błąd do konsoli przeglądarki „console.error(„Błąd sprawdzenia URL:”)”. Dzięki temu zostaje zapewniona odporność na błędy i stabilne działanie aplikacji w przypadku problemów z komunikacją między witryną a serwerem.

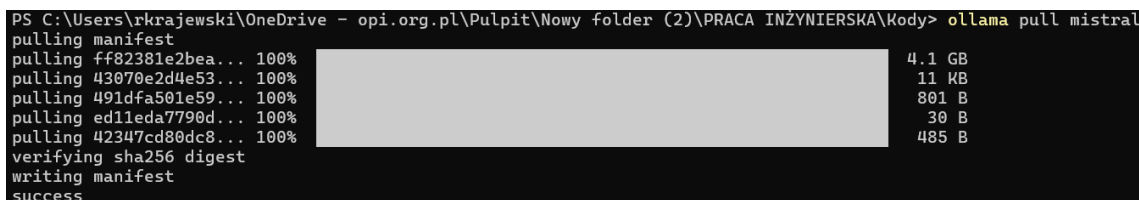
## 2.2 Instalacja i konfiguracja środowiska



```
Windows PowerShell
PS C:\Users\rkrajewski\OneDrive - opi.org.pl\Pulpit\Nowy folder (2)\PRACA INŻYNIERSKA\Kody> node server.js
✓ Serwer działa na http://localhost:3000
```

Rysunek 26. Uruchomienie serwera w PowerShell - Opracowanie: Rafał Krajewski

Zgodnie z rysunkiem 26 w celu uruchomienia aplikacji konieczna jest instalacja środowiska Node.js(v22.14.0), który umożliwi działanie lokalnego serwera. Po instalacji serwera, w katalogu „KODY” należy uruchomić następujące polecenie w PowerShellu: *npm install express cors body-parser natural nodemailer openai string-similarity*. Komenda ta instaluje wymagane biblioteki. Uruchomienie serwera odbywa się za pomocą komendy *node server.js*.

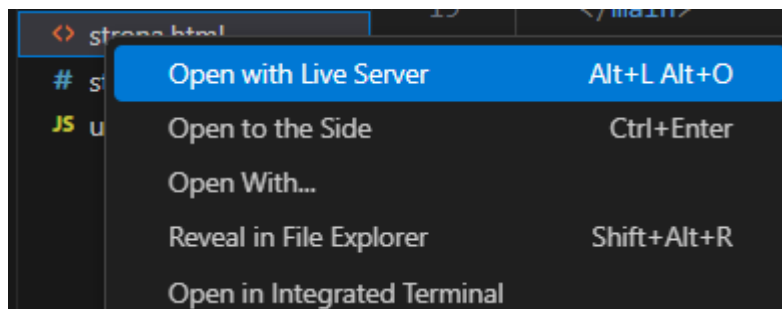


```
PS C:\Users\rkrajewski\OneDrive - opi.org.pl\Pulpit\Nowy folder (2)\PRACA INŻYNIERSKA\Kody> ollama pull mistral
pulling manifest
pulling ff82381e2bea... 100%
pulling 43070e2d4e53... 100%
pulling 491dfa501e59... 100%
pulling ed11eda7790d... 100%
pulling 42347cd80dc8... 100%
verifying sha256 digest
writing manifest
success
```

Rysunek 27. Pobranie modelu AI w PowerShell - Opracowanie: Rafał Krajewski

Zgodnie z rysunkiem 27 w celu umożliwienia generowania przykładowego maila phishingowego konieczna jest instalacja modelu AI Ollama (v0.6.1). Po instalacji modelu AI w katalogu „KODY”, należy uruchomić następujące polecenie w PowerShellu: *\$env:Path += ";C:\Program Files\Ollama\bin"*. Komenda ta dodaje tymczasowo nowy katalog do zmiennej „Path”. Następnie zgodnie z rysunkiem 27 pobieramy model AI

poleceniem: „ollama pull mistral”. Po każdorazowym zamknięciu terminala trzeba wykonać wyżej wymienione komendy ponownie.



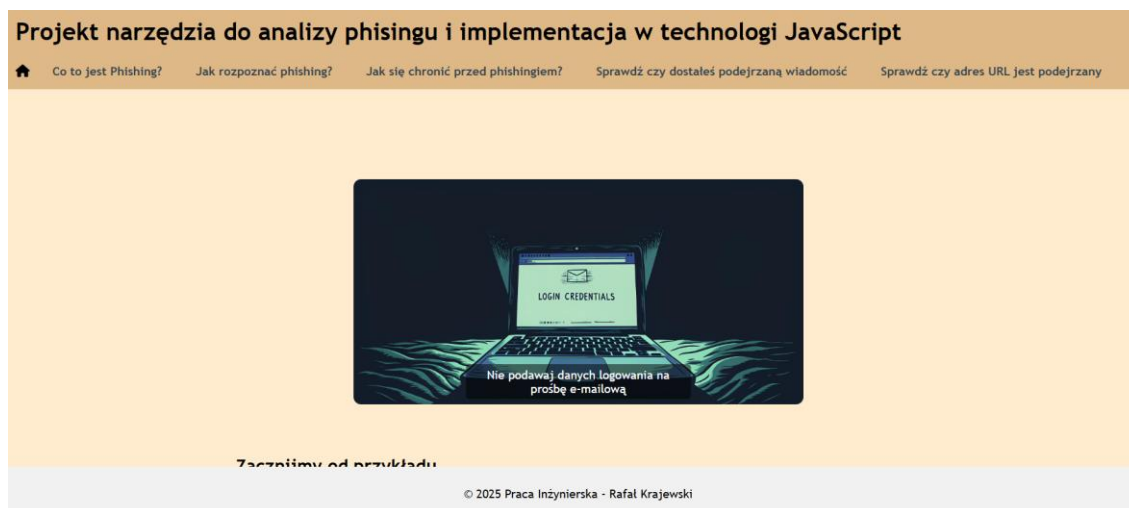
Rysunek 28. Uruchomienie strony internetowej poprzez wtyczkę Live Server - Opracowanie: Rafał Krajewski

Zgodnie z rysunkiem 28 w celu uruchomienia stron internetowych w przeglądarce konieczna jest instalacja wtyczki Live Server (autor: Ritwick Dey) do edytora kodu Visual Studio Code.. Aby wystartować witrynę internetową, po instalacji wtyczki klikamy prawym przyciskiem myszy np.: na „strona.html” i wybieramy opcję „*Open with Live Server*”. Zgodnie z rysunkiem 28 po kliknięciu, w przeglądarce uruchomi nam się wybrana przez nas strona internetowa.

## 3 Opis interfejsu użytkownika i testy aplikacji

### 3.1 Prezentacja interfejsu użytkownika

#### 3.1.1 Strona główna (strona.html)



Rysunek 29. Prezentacja graficzna strony startowej cz.1 - Opracowanie: Rafał Krajewski



Rysunek 30. Prezentacja graficzna strony startowej cz.2 - Opracowanie: Rafał Krajewski

Zgodnie z rysunkami 29 i 30 strona główna „start.html” na samej górze zawiera nawigację między podstronami. Jest ona wykorzystywana na każdej podstronie z osobna. Poniżej znajduje się komponent karuzeli „carousel” wyświetlający komunikaty edukacyjne dotyczące cyberbezpieczeństwa. Każdy slajd zawiera grafikę oraz podpis

informacyjny. Użytkownik może przełączać slajdy ręcznie lub automatycznie. Strona pełni funkcję wprowadzającą do tematu pracy inżynierskiej.

### 3.1.2 Informacje o phishingu (phi.html)



Rysunek 31. Prezentacja graficzna strony internetowej o rodzajach phishingu - Opracowanie: Rafał Krajewski

Zgodnie z rysunkiem 31 strona „phi.html” zawiera opis zjawiska phishingu oraz jego najczęściej występujące odmiany. Informacje te są przedstawione w formie listy wyróżnionymi nazwami typów ataków. Strona pełni funkcję edukacyjną i stanowi podstawę zrozumienia dalszych treści.

### 3.1.3 Jak rozpoznać phishing (rozpoznaniephi.html)



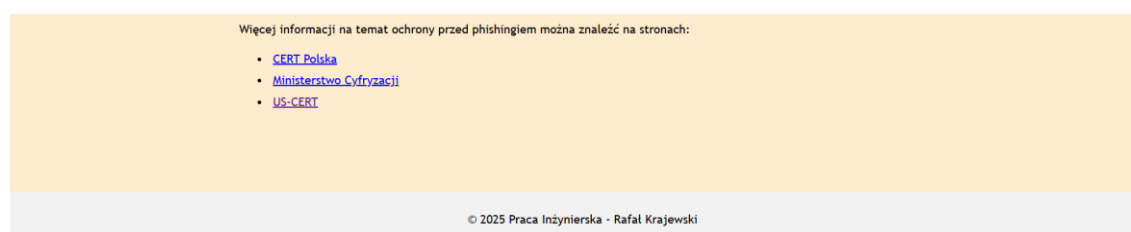
Rysunek 32. Prezentacja graficzna strony internetowej o rozpoznawaniu phishingu - Opracowanie: Rafał Krajewski

Na stronie „rozpoznaniephi.html” zgodnie z rysunkiem 32 przedstawiono wskazówki które pomagają użytkownikowi rozpoznać podejrzaną wiadomość. Treść została zaprezentowana w formie listy z wyróżnionymi kluczowymi elementami. Strona zawiera konkretne przykłady zachowań wskazujących na próbę oszustwa.

### 3.1.4 Jak się chronić przed phishingiem (ochronaphi.html)



Rysunek 33. Prezentacja graficzna strony internetowej o ochronie przed phishingiem - Opracowanie: Rafał Krajewski



Rysunek 34. Prezentacja graficzna strony internetowej o ochronie przed phishingiem - Opracowanie: Rafał Krajewski

Na stronie „ochronaphi.html” zgodnie z rysunkami 33 i 34 przedstawiono praktyczne metody ochrony przed phishingiem. Treść została zaprezentowana w formie listy uporządkowanej, w której przedstawiono kilka podstawowych zasad bezpieczeństwa. Do każdej z nich dołączono krótki opis. Na końcu strony umieszczono odnośniki do zewnętrznych, państwowych źródeł gdzie można wyszukać informacje na temat phishingu.

### 3.1.5 Analiza treści e-mail (sprawdzeniephi.html)



Rysunek 35. Prezentacja graficzna strony internetowej do analizy treści e-mail - Opracowanie: Rafał Krajewski

Zgodnie z rysunkiem 35 strona „sprawdzeniephi.html” zawiera formularz umożliwiający wprowadzenie podejrzanej treści e-mail. Po kliknięciu przycisku „Sprawdź” dane są przesyłane do serwera, który analizuje podobieństwo treści do zdefiniowanych wzorców phishingu. Wynik analizy w postaci komunikatu i procentowej oceny podobieństwa jest wyświetlany bezpośrednio pod formularzem.

### 3.1.6 Analiza adresu URL (sprawdzenieurl.html)



Rysunek 36. Prezentacja graficzna strony internetowej do analizy adresu URL - Opracowanie: Rafał Krajewski



Na stronie „sprawdzenieurl.html” zgodnie z rysunkiem 36 umożliwiono sprawdzenie, czy podany adres URL może być potencjalnie niebezpieczny. Użytkownik wpisuje adres w polu formularza, a następnie przesyłany jest on do serwera, który porównuje go z listą podejrzanych końcówek domen oraz legalnych adresów. Wynik analizy w postaci komunikatu i procentowej oceny podobieństwa jest wyświetlany bezpośrednio pod formularzem.

## **3.2 Testy i analiza wyników**

### **3.2.1 Metodyka testowania**

Celem przeprowadzonych testów było zweryfikowanie poprawności działania funkcji aplikacji analizującej potencjalne ataki phishingowe. Testy funkcjonalne miały na celu sprawdzenie zgodności działania systemu z wymaganiami funkcjonalnymi określonymi na etapie projektowania.

Do testowania zastosowano podejście czarnoskrzynkowe, co oznacza że testujący nie miał wglądu do kodu źródłowego analizowanych funkcji, koncentrując się na wpisywaniu danych i oczekiwaniu na wynik zgodnie z sylabusem ISTQB (Międzynarodowa Organizacja Certyfikująca Testerów Oprogramowania)[29]. Testy przeprowadzono ręcznie w przeglądarce internetowej.

Środowisko testowe:

- System operacyjny Windows 11 (build 22631.5189)
- Przeglądarka Google Chrome (v136.0.7103.114)
- Serwer lokalny Node.js (v22.14.0)
- Model AI Ollama (v0.6.1)
- Edytor Visual Studio Code (v1.98.2)
- Wtyczka Live Server (v5.7.9)
- Biblioteka natural (v8.0.1)
- Biblioteka body-parser (v2.2.0)
- Biblioteka cors (v2.8.5)
- Biblioteka express (v4.21.2)

- Biblioteka nodemailer (v6.10.0)
- Biblioteka openai (v4.85.2)
- Biblioteka string-similarity (v4.0.4)

Każdy przypadek testowy został zdefiniowany według jednolitego wzorca: identyfikator testu, nazwa scenariusza, dane wejściowe, oczekiwany rezultat, rzeczywisty wynik oraz status testu.

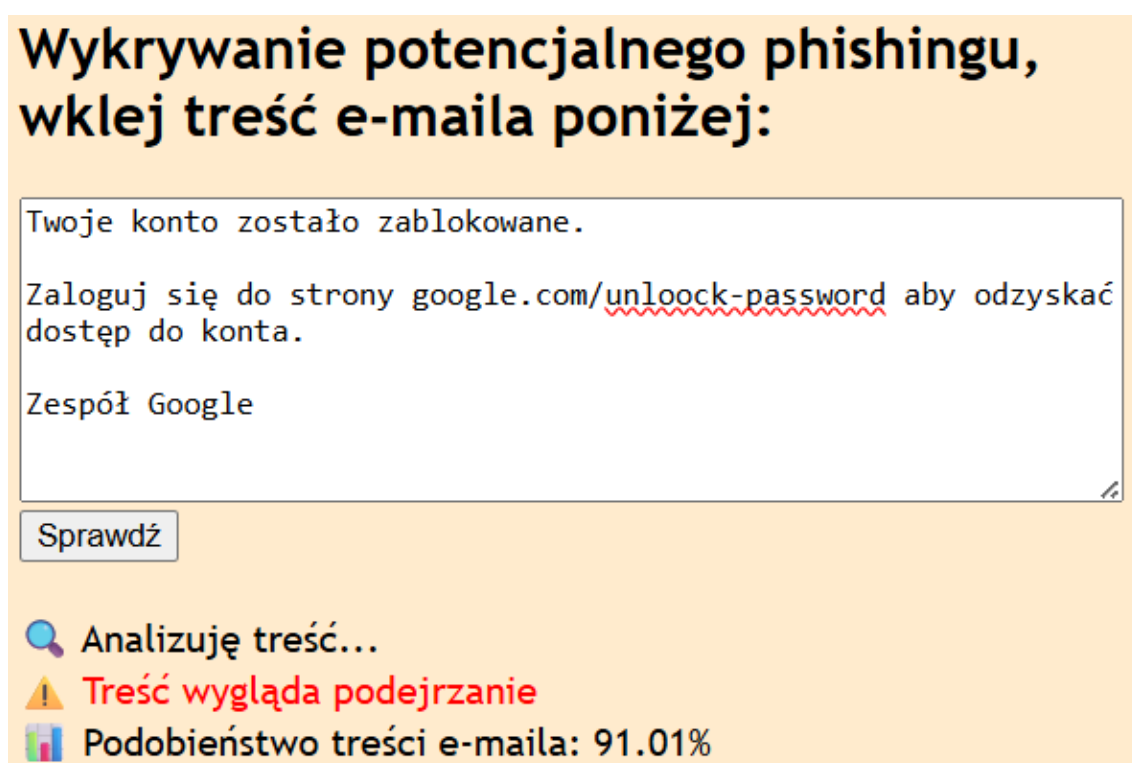
### 3.2.2 Scenariusze testowe

Poniżej przedstawiono zestaw wybranych scenariuszy testowych pokrywających kluczowe funkcjonalności aplikacji takie jak: analiza treści e-mail, sprawdzenie adresów URL oraz reakcja systemu na błędy.

Nr testu	Nazwa scenariusza	Dane wejściowe	Oczekiwany wynik	Rzeczywisty wynik	Status
1	Analiza e-mail z phishingiem	patrz: <a href="#">Rysunek 37</a>	Treść wygląda podejrzanie	Zgodny	Test zakończony pomyślnie
2	Analiza bezpiecznego e-maila	patrz: <a href="#">Rysunek 38</a>	Treść wygląda bezpiecznie	Zgodny	Test zakończony pomyślnie
3	Sprawdzenie podejrzanego adresu URL	patrz: <a href="#">Rysunek 39</a>	Adres URL wygląda podejrzanie:	Zgodny	Test zakończony pomyślnie
4	Sprawdzenie bezpiecznego adresu URL	patrz: <a href="#">Rysunek 40</a>	Adres URL wygląda bezpiecznie:	Zgodny	Test zakończony pomyślnie
5	Brak połączenia z serwerem	Wyłączony serwer lokalny, patrz: <a href="#">Rysunek 41</a>	Błąd połączenia z serwerem	Zgodny	Test zakończony pomyślnie

6	Wysłanie testowego e-maila phishing	patrz: <a href="#">Rysunek 42</a> , <a href="#">Rysunek 43</a> , <a href="#">Rysunek 44</a>	Dostarczenie testowego maila phishingowego	Zgodny	Test zakończony pomyślnie
7	Brak wysłania testowego e-maila phishing	Patrz: <a href="#">Rysunek 45</a>	Błąd podczas wysyłania testowego e-maila phishingu	Zgodny	Test zakończony pomyślnie

Tabela 1. Scenariusze testowe - Opracowanie: Rafał Krajewski



Rysunek 37. Analiza treści e-mail cz.1- Opracowanie: Rafał Krajewski

## Wykrywanie potencjalnego phishingu, wklej treść e-maila poniżej:

Cześć Studentko! Cześć Studencie!

Końcówka semestru, początek wakacji... a my mamy dla Ciebie porcję praktycznych wydarzeń, konsultacji i ogłoszeń, które pomogą Ci wejść na rynek pracy z większym luzem i planem! 📧

🎯 Nie wiesz co dalej? Zgłoś się na czerwcową konsultację! Kończysz studia i nie masz jeszcze planu? A może masz kilka pomysłów, ale trudno Ci się zdecydować? Przez cały czerwiec zapraszamy na indywidualne konsultacje kariery dla studentów ostatniego roku (każdego kierunku).

- ✅ Rozmowa o możliwościach po studiach
- ✅ Pomoc w CV i LinkedIn
- ✅ Przygotowanie do rozmów rekrutacyjnych
- ✅ Zbieranie myśli i uporządkowanie planów

- 📍 Online lub stacjonarnie – jak Ci wygodniej
- 📅 Terminy ustalane indywidualnie
- 📧 Zapisy mailowo na: [marta.jasiorowska@uth.edu.pl](mailto:marta.jasiorowska@uth.edu.pl)

Sprawdź

🔍 Analizuję treść...

✅ Treść wygląda bezpiecznie


📊 Podobieństwo treści e-maila: 59.81%


Rysunek 38. Analiza treści e-mail cz.2 - Opracowanie: Rafał Krajewski


## Wykrywanie podejrzanego adresu URL, wklej go poniżej:

<http://mail.google.com.ru>

Sprawdź

 Analizuję treść...

 **Adres URL wygląda podejrzanie:**  
[mail.google.com.ru](http://mail.google.com.ru)


 Podobieństwo domeny URL do zaufanych: 73.52%


Rysunek 39. Analiza adresu URL cz.1 - Opracowanie: Rafał Krajewski


## Wykrywanie podejrzanego adresu URL, wklej go poniżej:

<https://www.bankmillennium.pl>

Sprawdź

 Analizuję treść...

 **Adres URL wygląda bezpiecznie:**  
[www.bankmillennium.pl](https://www.bankmillennium.pl)

 Podobieństwo domeny URL do zaufanych: 73.20%

Rysunek 40. Analiza adresu URL cz.2 - Opracowanie: Rafał Krajewski

## Wykrywanie potencjalnego phishingu, wklej treść e-maila poniżej:

Wklej e-mail tutaj

Sprawdź



Analizuję treść...



Błąd połączenia z serwerem.



Treść wygląda bezpiecznie



Podobieństwo treści e-maila: 0.00%

Rysunek 41. Weryfikacja mechanizmu detekcji błędów - Opracowanie: Rafał Krajewski

## Zaczniemy od przykładu...

Wpisz poniżej swojego maila, a wyśle do Ciebie przykładowy mail phishing.

r.krajewski26@gmail.com

Wyślij testowy phishing

Rysunek 42. Wysłanie testowego maila phishingowego cz.1 - Opracowanie: Rafał Krajewski

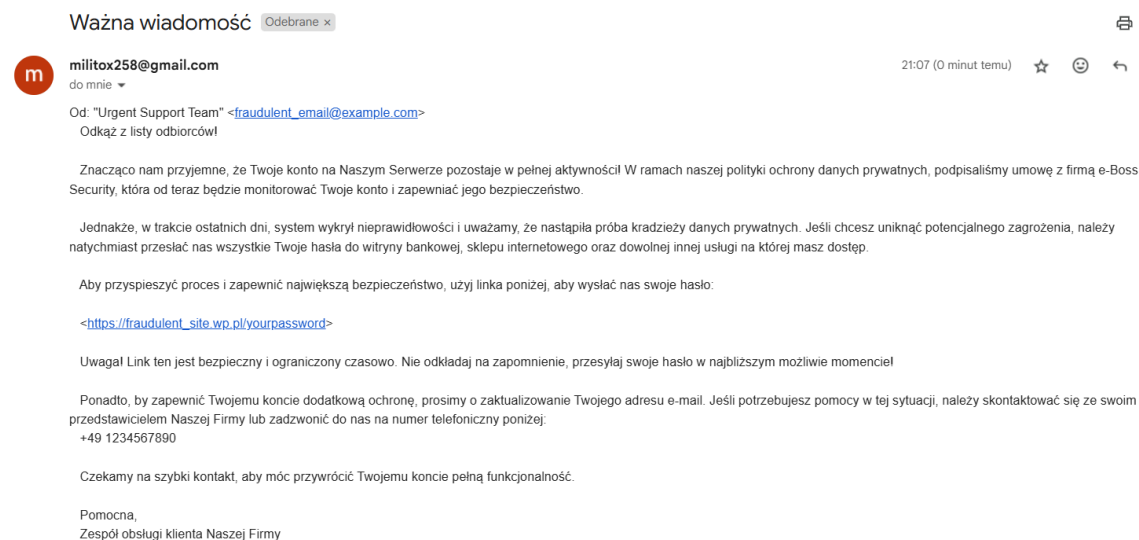
## Komunikat ze strony 127.0.0.1:5500



E-mail został wysłany!

OK

Rysunek 43. Wysłanie testowego maila phishingowego cz.2 - Opracowanie: Rafał Krajewski



Rysunek 44. Wysłanie testowego maila phishingowego cz.3 - Opracowanie: Rafał Krajewski

Komunikat ze strony 127.0.0.1:5500

✗ Wystąpił błąd podczas wysyłania phishingu.



Rysunek 45. Błąd podczas wysyłania testowego phishingu - Opracowanie: Rafał Krajewski

### 3.3 Analiza wyników testów

Testy funkcjonalne przeprowadzone na opracowanym systemie wykazały pełną zgodność działania w odniesieniu do oczekiwań. Skupiono się na weryfikacji kluczowych komponentów, takich jak analiza treści wiadomości e-mail, detekcja podejrzanego adresu URL, walidacja błędów w odniesieniu do połączenia z serwerem i wysłania e-maila phishingowego oraz wysłanie testowego e-maila phishingowego. Wszystkie wyżej wymienione funkcjonalności działały poprawnie, bez względu na formę danych wejściowych.

Zaobserwowano, że:

- **System skutecznie wykrywał charakterystyczne cechy phishingu:**

Moduł analizujący treść e-maili (plik „phishingrozpoznanie.js”) potrafił rozpoznać alarmujące sformułowania typowe dla phishingu. W przypadku wiadomości zawierających treści zbliżone do zdefiniowanych wzorców phishingowych, system wskazywał procentową zgodność, co umożliwiała użytkownikowi samodzielną ocenę ryzyka. W testach wykazano wysoką skuteczność detekcji przy minimalnej liczbie błędnych klasyfikacji,

- **Detekcja podejrzanych adresów URL działała z wysoką dokładnością:**

Moduł analizujący adres URL (plik „urlrozpoznanie.js”) poprawnie identyfikował niebezpieczne adresy, w tym te zawierające podejrzane końcówki domen (np. .tk, .ru, .ml) oraz te z dużym podobieństwem do znanych legalnych domen (np. facebook.com). Zastosowanie algorytmu Jaro-Winklera pozwoliło na wykrywanie tzw. „typo-squattingu” – fałszywych domen podszywających się pod znane marki. Wyniki prezentowane użytkownikowi były zrozumiałe i zawierały szczegółowe dane procentowe,

- **Obsługa błędów połączenia z serwerem była stabilna i przejrzysta:**

System reagował prawidłowo na błędy połączenia z serwerem. W przypadku przerwania komunikacji lub błędnego formatu danych wejściowych, użytkownik otrzymywał czytelny komunikat, a aplikacja nie zawieszała się. Obsługa błędów została zrealizowana zarówno po stronie klienta, jak i serwera, co zwiększało odporność całego systemu,

- **Wysyłanie testowego e-maila phishingowego i obsługa błędów działała poprawnie:**

Test funkcji wysyłania przykładowych e-maili phishingowych, potwierdził, że system generuje i przesyła wiadomości zawierające symulowane elementy phishingowe. Wysyłka działała po poprawnej konfiguracji modułu nodemailer i modelu AI (Ollama).



Użytkownik otrzymywał komunikat zwrotny o powodzeniu operacji lub opis błędu, co było szczególnie przydatne w przypadku testowania z nieprawidłową konfiguracją,

- **System radził sobie z różnorodnymi danymi wejściowymi:**

Zarówno analiza e-maili jak i URL była odporna na puste pola, brak nagłówków czy niecodzienne formatowanie tekstu. Dzięki odpowiednim mechanizmom walidacji i obsługi wyjątków, system nie przerywał pracy nawet w przypadku niekompletnych lub źle sformatowanych danych, co jest kluczowe w aplikacjach dostępnych dla użytkowników końcowych,

- **Interfejs użytkownika był intuicyjny, responsywny i przejrzysty:**

Dzięki wykorzystaniu dynamicznego ładowania sekcji HTML i dopracowanego wyglądu strony, interfejs był przejrzysty i funkcjonalny. Formularze reagowały dynamicznie na akcje użytkownika, a komponenty takie jak karuzela z komunikatami edukacyjnymi pozytywnie wpływały na doświadczenie użytkownika.

### 3.4 Ocena wydajności i stabilności

W ramach oceny wydajności i stabilności systemu przeprowadzono wyżej opisane testy mające na celu sprawdzenie szybkości działania, poprawności działania oraz odporności na błędy. Badano czas reakcji serwera na podstawowe operacje oraz zachowanie interfejsu użytkownika przy typowych interakcjach. Uzyskane wyniki potwierdzają wysoką wydajność i stabilność systemu:

- Czas przetwarzania treści e-mail wyniósł średnio 15 milisekund, co zapewnia użytkownikowi szybki zwrot informacji,
- Czas analizy adresu URL wyniósł średnio 10 milisekund, dzięki wykorzystaniu algorytmu Jaro-Winklera,
- System utrzymywał wysoką stabilność, bez spadków wydajności, błędów serwera czy zatrzymań pracy.

Obsługa formularzy oraz interfejs użytkownika działały płynnie i bez zakłóceń. Wszystkie komponenty HTML, w tym przyciski, pola tekstowe i formularze, reagowały zgodnie z oczekiwaniami użytkownika, bez opóźnień, błędów czy konieczności ponownego ładowania strony. Dzięki zastosowaniu technik dynamicznego ładowania treści, zawartość stron była aktualizowana natychmiast po wykonaniu odpowiednich akcji, co znacząco poprawiało komfort korzystania z aplikacji. Co istotne, nawet w przypadku wprowadzenia niekompletnych lub błędnych danych wejściowych, takich jak puste pola, niepoprawny format adresu e-mail czy nieczytelna treść wiadomości, system pozostawał w pełni stabilny. Aplikacja nie zawieszała się i nie traciła połączenia z serwerem, a użytkownik otrzymywał jasne, kontekstowe komunikaty informujące o błędzie i sugerujące sposób poprawy. Takie podejście zwiększało nie tylko niezawodność działania systemu, ale również jego użyteczność oraz dostępność dla osób mniej zaawansowanych technicznie.

## **4 Wnioski i perspektywy rozwoju**

### **4.1 Podsumowanie pracy**

Celem niniejszej pracy inżynierskiej było zaprojektowanie oraz implementacja narzędzia do analizy potencjalnych ataków phishingowych. W ramach realizacji celu stworzono stronę internetową, na której umieszczono aplikację webową opartą na technologii JavaScript, działającą z wykorzystaniem środowiska Node.js. Kluczowym komponentem systemu była biblioteka „natural”, służąca do przetwarzania języka naturalnego, co umożliwiło analizę treści wiadomości e-mail oraz adresów URL. Dodatkowo umożliwiono użytkownikowi wygenerowanie testowego e-maila phishingowego przy użyciu sztucznej inteligencji, za pomocą modułu Ollama w sposób zautomatyzowany i realistyczny.

Opracowanie rozwiązania umożliwia użytkownikowi samodzielne przeprowadzenie wstępnej analizy podejrzanych wiadomości i linków, wykrywając typowe cechy ataków phishingowych. Zaimplementowane algorytmy dają możliwość porównania treści do zdefiniowanych wzorców phishingu, a także porównać wpisany adres URL do listy zaufanych domen i sprawdzić, czy nie zawiera on podejrzanych końcówek adresu. W dodatku użytkownik ma możliwość zaobserwować jak wygląda e-mail phishingowy.

Testy funkcjonalne potwierdziły skuteczność narzędzia detekcji zagrożeń w różnych scenariuszach, a system podczas ich przeprowadzania działał stabilnie i wydajnie. Aplikacja została wzbogacona o prosty i intuicyjny interfejs użytkownika, co dodatkowo podnosi jej użyteczność.

### **4.2 Ocena osiągnięcia celów pracy**

Wszystkie założone cele pracy zostały w pełni zrealizowane. System funkcjonuje zgodnie z wymaganiami funkcjonalnymi i technicznymi, zapewniając skuteczne wykrywanie potencjalnych zagrożeń phishingowych. Wdrożony algorytm porównywania treści e-maila oraz weryfikacja adresu URL umożliwiają skuteczne wykrywanie typowych zagrożeń, a także generowanie i wysyłkę testowych wiadomości phishingowych, co stanowi wartość dodaną w kontekście edukacyjnym. Aplikacja

charakteryzuje się prostotą obsługi, przejrzystym interfejsem oraz szybkim czasem odpowiedzi, co czyni ją dostępną nawet dla mniej zaawansowanych użytkowników.

Dodatkowym osiągnięciem projektu było wdrożenie podstawowej obsługi błędów oraz automatyczna reakcja na nieprawidłowe dane wejściowe, co zapewniło stabilność działania i zwiększyło bezpieczeństwo aplikacji.

### **4.3 Możliwości dalszego rozwoju systemu**

Zaprojektowane narzędzie stanowi solidny fundament, który w przyszłości może zostać rozbudowany o dodatkowe możliwości zwiększające jego użyteczność i skuteczność.

Propozycje kierunków rozwoju obejmują m.in.:

- Analizę załączników w wiadomościach e-mail, z możliwością wykrywania złośliwych plików lub ukrytych skryptów,
- Integrację z zewnętrznymi bazami danych zagrożeń, co umożliwi automatyczne sprawdzanie domen i adresów URL pod kątem znanych zagrożeń,
- Wielojęzyczną analizę treści e-mail np.: angielskim, hiszpańskim czy niemieckim,
- Optymalizację współpracy algorytmów z AI, np.: poprzez wybór stylu wiadomości phishingowej (fałszywa faktura, powiadomienie o logowaniu, wiadomość z banku itp.),
- Zwiększenie skalowalności i wdrożenie w chmurze, co umożliwiłoby korzystanie z systemu przez wielu użytkowników jednocześnie.

### **4.4 Potencjalne zastosowania praktyczne**

Zaprojektowane narzędzie może znaleźć zastosowanie w wielu obszarach, zarówno edukacyjnych, jak i komercyjnych m.in:

- Instytucje edukacyjne, jako interaktywne narzędzie do szkoleń z zakresu cyberbezpieczeństwa,

- Firmy i organizacje, jako wsparcie działów IT i cyberbezpieczeństwa
- Projekty open source, jako otwarte i elastyczne narzędzie ostrzegające użytkownika przed zagrożeniami

## 5 Bibliografia

- [1] Oracle, „Przyspiesz swoje operacje dzięki technologii IoT”. Dostęp: 9 czerwiec 2025. [Online]. Dostępne na: <https://www.oracle.com/pl/internet-of-things/>
- [2] T. Kozon, „JavaScript – Skryptowy język programowania do tworzenia aplikacji”. Dostęp: 9 czerwiec 2025. [Online]. Dostępne na: <https://boringowl.io/tag/javascript>
- [3] Semcore, „Adres URL - co to jest i z jakich elementów się składa? | Semcore”, Adres URL – co to jest i z jakich elementów się składa? Dostęp: 9 czerwiec 2025. [Online]. Dostępne na: <https://semcore.pl/url-co-to-jest-jak-go-znalezc-i-z-jakich-elementow-sie-sklada/>
- [4] „Cyberbezpieczeństwo.pdf”. Dostęp: 9 czerwiec 2025. [Online]. Dostępne na: [https://piwparczew.pl/bip/328\\_piwparczew/fckeditor/file/cyberbezpieczenstwo/Cyberbezpiecze%C5%84stwo.pdf?utm\\_source=chatgpt.com](https://piwparczew.pl/bip/328_piwparczew/fckeditor/file/cyberbezpieczenstwo/Cyberbezpiecze%C5%84stwo.pdf?utm_source=chatgpt.com)
- [5] T. Kozon, „Node.js, czyli narzędzie do uruchamiania JavaScript”. Dostęp: 9 czerwiec 2025. [Online]. Dostępne na: <https://boringowl.io/tag/nodejs>
- [6] A. T. Carneiro, „Understanding Phishing Attacks: Prevention and Awareness”, Smart Tech Journal. Dostęp: 9 czerwiec 2025. [Online]. Dostępne na: <https://smarttechjournal.com/understanding-phishing-attacks-prevention-and-awareness>
- [7] J. Jancelewicz, „Phishing i pokrewne ataki socjotechniczne jako zagrożenie dla organizacji pozarządowych”, *Kwart. 3 Sekt.*, t. 5960, nr 34, s. 78–88, 2022, doi: 10.26368/17332265-59/60-3/4-2022-5.
- [8] M. Bitaab i in., „Scam Pandemic: How Attackers Exploit Public Fear through Phishing”, 23 marzec 2021, *arXiv*: arXiv:2103.12843. doi: 10.48550/arXiv.2103.12843.
- [9] S. Grzebielec, „Analysis of the vulnerability of IT system users to a phishing attack”, *J. Comput. Sci. Inst.*, t. 15, s. 164–167, cze. 2020, doi: 10.35784/jcsi.2049.
- [10] M. Grzelak i K. Liedel, „Bezpieczeństwo w cyberprzestrzeni. Zagrożenia i wyzwania dla Polski – zarys problemu”, 2012.
- [11] A. Warchoń, „Pojęcie cyberprzestrzeni w strategiach bezpieczeństwa państw członkowskich Unii Europejskiej”, *Ann. Univ. Paedagog. Cracoviensis Stud. Secur.*, 2019.
- [12] T. Dębowski, Red., *Cyberbezpieczeństwo wyzwaniem XXI wieku*. Wrocław ; Łódź: Wydawnictwo Naukowe ArchaeGraph Diana Łukomiak, 2018.
- [13] K. Chałubińska-Jentkiewicz, „Cyberbezpieczeństwo – zagadnienia definicyjne”, *Cybersecurity Law*, t. 2, nr 2, s. 7–23, mar. 2021, doi: 10.35467/cal/133828.
- [14] A. Pieczywok, M. Kościelny, i S. Wasilewski, „Kształtowanie bezpieczeństwa środowiskowego i w cyberprzestrzeni – porównanie zagrożeń”, *Cybersecurity Law*, t. Vol. 6 (2), 2021, doi: 10.35467/cal/146459.
- [15] „Cyberterroryzm jako szczególna forma terroryzmu w kontekście zagrożenia bezpieczeństwa infrastruktury krytycznej na przykładzie sektora energetycznego. Od historii do współczesności”. Dostęp: 9 czerwiec 2025. [Online]. Dostępne na: [https://d1wqtxts1xzle7.cloudfront.net/103609422/PC\\_2021\\_20K\\_20F\\_20i\\_20B-libre.pdf?1687347819=&response-content-disposition=inline%3B+filename%3DKultura\\_Fizyczna\\_I\\_Bezpieczenstwo\\_Wybran.pdf&Expires=1749426769&Signature=YytnHAr58QKoj1bgFmYxCdbGsYWv3w-8B9i8Alc9KAPrvs4PnNy7QclF-24ECdUzG5tEPrJjckdF5Gidk8FUVKBhI4hrR1ajYo13RRqgccEHsIVfaLPBkqPN8Mgnzo7ipjI0D0fLVc0vPsALMqnzfV~3lRUP6WmJCKMuHfNWz4XAgmwvbaB~](https://d1wqtxts1xzle7.cloudfront.net/103609422/PC_2021_20K_20F_20i_20B-libre.pdf?1687347819=&response-content-disposition=inline%3B+filename%3DKultura_Fizyczna_I_Bezpieczenstwo_Wybran.pdf&Expires=1749426769&Signature=YytnHAr58QKoj1bgFmYxCdbGsYWv3w-8B9i8Alc9KAPrvs4PnNy7QclF-24ECdUzG5tEPrJjckdF5Gidk8FUVKBhI4hrR1ajYo13RRqgccEHsIVfaLPBkqPN8Mgnzo7ipjI0D0fLVc0vPsALMqnzfV~3lRUP6WmJCKMuHfNWz4XAgmwvbaB~)

- 7ZrUsGoHgFhkBnYXbxJKpRi5WaA1BCeu4AvHT6CiPP2Vg26cg3adww0bpDoAkJOcNUMzqibJ1NXzEF9J5bIOutiPGMCSXRcvel1NdmgyTonuVoGFrOL6PgVl-MneaJykTaCOfeKLvQYRVWob7I8y7Xb2-5pwXFWeg\_\_&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA#page=158
- [16] „Backend – co to jest i o czym wiedzieć? | Semcore”. Dostęp: 21 maj 2025. [Online]. Dostępne na: <https://semcore.pl/backend-co-to-jest-i-o-czym-wiedziec/>
  - [17] 1stplace, „Czym jest Div? Odkrywamy świat HTML-owych kontenerów • 1stplace.pl”. Dostęp: 9 czerwiec 2025. [Online]. Dostępne na: <https://1stplace.pl/blog/czym-jest-div-odkrywamy-swiat-html-owych-kontenerow/>
  - [18] Shebang, „Atrybut HTML class”, Atrybut class. Dostęp: 9 czerwiec 2025. [Online]. Dostępne na: <https://shebang.pl/html/atribut-class/>
  - [19] E. A. Meyer i A. Trojan, *CSS: kaskadowe arkusze stylów: przewodnik encyklopedyczny*. Gliwice: Wydawnictwo Helion, 2008.
  - [20] „Find Icons with the Perfect Look & Feel | Font Awesome”. Dostęp: 10 czerwiec 2025. [Online]. Dostępne na: <https://fontawesome.com/icons>
  - [21] W3C, „HTML Standard”, HTML Living Standard — Last Updated 9 June 2025. Dostęp: 10 czerwiec 2025. [Online]. Dostępne na: <https://html.spec.whatwg.org/multipage/dom.html#semantics-2>
  - [22] „Responsywna strona internetowa – Poradnik | Semcore”. Dostęp: 10 czerwiec 2025. [Online]. Dostępne na: <https://semcore.pl/responsywna-strona-internetowa-poradnik/>
  - [23] „Czym są meta tagi? | Semcore”. Dostęp: 10 czerwiec 2025. [Online]. Dostępne na: <https://semcore.pl/sloownik/meta-tag/>
  - [24] „Asynchroniczność JS na stronie – jak działa? | Semcore”. Dostęp: 10 czerwiec 2025. [Online]. Dostępne na: <https://semcore.pl/asynchronicznosc-js-na-stronie-jak-dziala/>
  - [25] IBM, „What Is an API Endpoint? | IBM”. Dostęp: 10 czerwiec 2025. [Online]. Dostępne na: <https://www.ibm.com/think/topics/api-endpoint>
  - [26] „How to use Tokenizer in JavaScript?”, GeeksforGeeks. Dostęp: 10 czerwiec 2025. [Online]. Dostępne na: <https://www.geeksforgeeks.org/how-to-use-tokenizer-in-javascript/>
  - [27] „Term Frequency-Inverse Document Frequency (TF-IDF) – co to jest i do czego służy? | Semcore”. Dostęp: 10 czerwiec 2025. [Online]. Dostępne na: <https://semcore.pl/sloownik/tf-idf/>
  - [28] „javascript07.pdf”. Dostęp: 22 maj 2025. [Online]. Dostępne na: <https://eugeniuszpiechula.pl/dokumenty/javascript07.pdf>
  - [29] „CTFL\_2018\_Sloownik\_wyrazen\_zwiazanych\_z\_testowaniem\_wykorzystywanych\_w\_sylabusie.pdf”. Dostęp: 11 czerwiec 2025. [Online]. Dostępne na: [https://b2bnetwork.pl/wp-content/uploads/2020/04/CTFL\\_2018\\_Sloownik\\_wyrazen\\_zwiazanych\\_z\\_testowaniem\\_wykorzystywanych\\_w\\_sylabusie.pdf?utm\\_source=chatgpt.com](https://b2bnetwork.pl/wp-content/uploads/2020/04/CTFL_2018_Sloownik_wyrazen_zwiazanych_z_testowaniem_wykorzystywanych_w_sylabusie.pdf?utm_source=chatgpt.com)

## 6 Spis rysunków i tabel

Rysunek 1. Struktura kodu źródłowego - Opracowanie: Rafał Krajewski .....	11
Rysunek 2. Zawartość folderu img - Opracowanie: Rafał Krajewski .....	13
Rysunek 3. Struktura folderu node_modules cz.1 - Opracowanie: Rafał Krajewski .....	14
Rysunek 4. Struktura folderu node_modules cz.2 - Opracowanie: Rafał Krajewski .....	15
Rysunek 5. Struktura komponentu karuzeli w HTML z komunikatami dotyczącymi cyberbezpieczeństwa – opracowanie: Rafał Krajewski.....	16
Rysunek 6. Struktura pliku ze zmiennymi w JavaScript - Opracowanie: Rafał Krajewski .....	17
Rysunek 7. Struktura pliku nawigacji w HTML - Opracowanie: Rafał Krajewski.....	18
Rysunek 8. Struktura pliku strony internetowej o ochronie przed phishingiem w HTML - Opracowanie: Rafał Krajewski.....	20
Rysunek 9. Struktura pliku konfiguracyjnego serwera node.js - Opracowanie: Rafał Krajewski .....	22
Rysunek 10. Struktura pliku strony internetowej o rodzajach phishingu w HTML - Opracowanie: Rafał Krajewski .....	23
Rysunek 11. Struktura pliku analizującego treść e-maila w JavaScript – Opracowanie: Rafał Krajewski .....	24
Rysunek 12. Struktura strony internetowej o rozpoznawaniu phishingu w HTML – Opracowanie: Rafał Krajewski .....	26
Rysunek 13. Struktura sekcji ze strony startowej w HTML - Opracowanie : Rafał Krajewski .....	27
Rysunek 14. Struktura pliku konfiguracyjnego serwera w JavaScript cz.1 - Opracowanie: Rafał Krajewski .....	29
Rysunek 15. Struktura pliku konfiguracyjnego serwera w JavaScript cz.2 - Opracowanie: Rafał Krajewski .....	30
Rysunek 16. Struktura pliku konfiguracyjnego serwera w JavaScript cz.3 - Opracowanie: Rafał Krajewski .....	31
Rysunek 17. Włączenie serwera Node.js - Opracowanie: Rafał Krajewski .....	34
Rysunek 18. Struktura pliku skryptowego w JavaScript cz.1 - Opracowanie: Rafał Krajewski .....	35
Rysunek 19. Struktura pliku skryptowego w JavaScript cz.2 - Opracowanie: Rafał Krajewski .....	36
Rysunek 20. Struktura pliku do analizy treści e-mail w HTML - Opracowanie : Rafał Krajewski .....	39
Rysunek 21. Struktura pliku do analizy adresu URL w HTML - Opracowanie : Rafał Krajewski .....	40
Rysunek 22. Struktura pliku witryny internetowej w HTML - Opracowanie: Rafał Krajewski .....	41
Rysunek 23. Struktura pliku reguł formatowania w CSS cz.1 - Opracowanie: Rafał Krajewski .....	43



Rysunek 24. Struktura pliku reguł formatowania w CSS cz.2- Opracowanie: Rafał Krajewski .....	44
Rysunek 25. . Struktura pliku do analizy adresu URL w JavaScript - Opracowanie : Rafał Krajewski .....	49
Rysunek 26. Uruchomienie serwera w PowerShell - Opracowanie: Rafał Krajewski ...	51
Rysunek 27. Pobranie modelu AI w PowerShell - Opracowanie: Rafał Krajewski .....	51
Rysunek 28. Uruchomienie strony internetowej poprzez wtyczkę Live Server - Opracowanie: Rafał Krajewski .....	52
Rysunek 29. Prezentacja graficzna strony startowej cz.1 - Opracowanie: Rafał Krajewski .....	53
Rysunek 30. Prezentacja graficzna strony startowej cz.2 - Opracowanie: Rafał Krajewski .....	53
Rysunek 31. Prezentacja graficzna strony internetowej o rodzajach phishingu - Opracowanie: Rafał Krajewski .....	54
Rysunek 32. Prezentacja graficzna strony internetowej o rozpoznawaniu phishingu - Opracowanie: Rafał Krajewski .....	54
Rysunek 33. Prezentacja graficzna strony internetowej o ochronie przed phishingiem - Opracowanie: Rafał Krajewski .....	55
Rysunek 34. Prezentacja graficzna strony internetowej o ochronie przed phishingiem - Opracowanie: Rafał Krajewski .....	55
Rysunek 35. Prezentacja graficzna strony internetowej do analizy treści e-mail - Opracowanie: Rafał Krajewski .....	56
Rysunek 36. Prezentacja graficzna strony internetowej do analizy adresu URL - Opracowanie: Rafał Krajewski .....	56
Tabela 1. Scenariusze testowe - Opracowanie: Rafał Krajewski.....	59
 Rysunek 37. Analiza treści e-mail cz.1- Opracowanie: Rafał Krajewski .....	59
Rysunek 38. Analiza treści e-mail cz.2 - Opracowanie: Rafał Krajewski .....	60
Rysunek 39. Analiza adresu URL cz.1 - Opracowanie: Rafał Krajewski.....	61
Rysunek 40. Analiza adresu URL cz.2 - Opracowanie: Rafał Krajewski.....	61
Rysunek 41. Weryfikacja mechanizmu detekcji błędów - Opracowanie: Rafał Krajewski .....	62
Rysunek 42. Wysłanie testowego maila phishingowego cz.1 - Opracowanie: Rafał Krajewski .....	62
Rysunek 43. Wysłanie testowego maila phishingowego cz.2 - Opracowanie: Rafał Krajewski .....	62
Rysunek 44. Wysłanie testowego maila phishingowego cz.3 - Opracowanie: Rafał Krajewski .....	63
Rysunek 45. Błąd podczas wysyłania testowego phishingu - Opracowanie: Rafał Krajewski .....	63