

# Sprawozdanie z laboratorium 1

Mikołaj Kubś 272662

14 kwietnia 2025

## 1 Cel zadania

Celem zadania było zapoznanie się z procesem tworzenia prostych stron Single Page Application (SPA), a także integracją z Azure Static Web i Github Pages. Dodatkowo wykorzystano technikę leniwego ładowania i mechanizm reCAPTCHA.

### 1.1 Wprowadzenie

Single Page Application (SPA) to rodzaj aplikacji internetowej, w której nawigacja odbywa się poprzez asynchroniczne ładowanie poszczególnych elementów strony, takich jak sekcje lub całe widoki, bez konieczności przeładowywania całej strony. Dzięki temu użytkownik doświadcza płynniejszej interakcji, ponieważ zmiany w interfejsie są natychmiastowe i nie wymagają pełnego odświeżenia przeglądarki.

W aplikacjach SPA cała zawartość jest zazwyczaj ładowana jednorazowo przy pierwszym wejściu na stronę, a późniejsze interakcje z użytkownikiem prowadzą do dynamicznej aktualizacji wyświetlanych danych za pomocą JavaScript. To podejście pozwala na szybsze działanie aplikacji oraz lepsze wykorzystanie zasobów sieciowych, ponieważ jedynie zmieniane elementy są przesyłane między serwerem a klientem.

### 1.2 Tworzenie aplikacji SPA z wykorzystaniem HTML i JS

#### 1.2.1 Kod HTML

Plik `index.html` definiuje prosty kod HTML aplikacji:

```

<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <meta http-equiv="X-UA-Compatible" content="ie=edge" />
    <title>SPA PIAC TEST</title>
    <link rel="stylesheet" href="style.css" />
  </head>
  <body>
    <header class="header">
      <ul class="Header-links-ul">
        <li class="header-link" id="about-link">About Me</li>
        <li class="header-link" id="contact-link">Contact</li>
        <button id="theme-toggle">Toggle Theme</button>
      </ul>
    </header>
    <main>
      <h1 class="title">Hello World!</h1>
      <p>
        Lorem Ipsum is simply dummy text of the printing and typesetting
        industry...
      </p>
    </main>
    <script src="js/router.js"></script>
  </body>
</html>

```

Rysunek 1: Kod index.html

Kod ten jest podmieniany przez `router.js`.

### 1.2.2 Stylizacja CSS

Plik `style.css` definiuje kaskadowe arkusze stylów:

```
html,
body {
  font-family: sans-serif;
  text-align: center;
  height: 100%;
  margin: 0;
  padding: 0;
  width: 100%;
}

header {
  display: flex;
  justify-content: space-around;
  align-items: center;
}

.Header-links-ul {
  width: 60%;
  list-style: none;
  display: flex;
  justify-content: space-around;
}

.header-link {
  padding: 0.4rem;
  border-radius: 2px;
  cursor: pointer;
}

.header-link:hover {
  border-bottom: 1px solid white;
}
```

### 1.2.3 Kod JavaScript

Plik `router.js` zarządza nawigacją w aplikacji SPA:

```
let pageUrls = {
  about: '/index.html?about',
  contact: '/index.html?contact',
  gallery: '/index.html?gallery'
};

function OnStartup() {
  popStateHandler();
}

OnStartup();

document.querySelector('#about-link').addEventListener('click', (event) => {
  let stateObj = { page: 'about' };
  document.title = 'About';
  history.pushState(stateObj, "about", "?about");
  RenderAboutPage();
});

document.querySelector('#gallery-link').addEventListener('click', (event) => {
  let stateObj = { page: 'gallery' };
  document.title = 'Gallery';
  history.pushState(stateObj, "gallery", "?gallery");
  RenderGalleryPage();
});

document.querySelector('#contact-link').addEventListener('click', (event) => {
  let stateObj = { page: 'contact' };
  document.title = 'Contact';
  history.pushState(stateObj, "contact", "?contact");
  RenderContactPage();
});

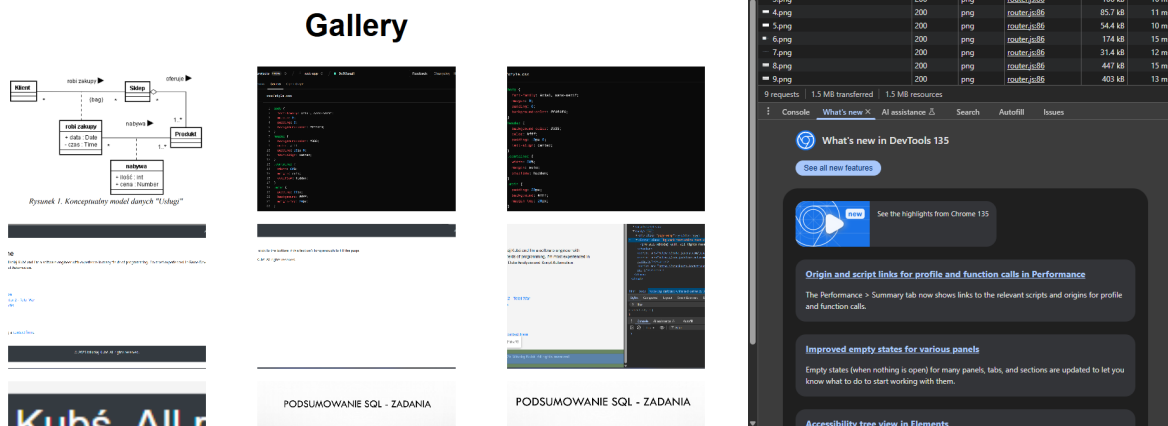
function RenderGalleryPage() {
  document.querySelector('main').innerHTML = `
    <br />
    <br />
    <br />
    <br />
    <br />
    <br />
    <br />
    <br />
    <br />
    <br />
  `;
}
```

Rysunek 3: Fragment kodu `router.js`

### 1.2.4 Leniwe ładowanie

Do implementacji leniwego ładowania obrazków wykorzystano Intersection Observer API. Obrazki są tworzone dynamicznie, a atrybut `src` jest przypisany dopiero, gdy obrazki powinny być widoczne.





Rysunek 5: Galeria w aplikacji

### 1.2.5 Implementacja reCAPTCHA

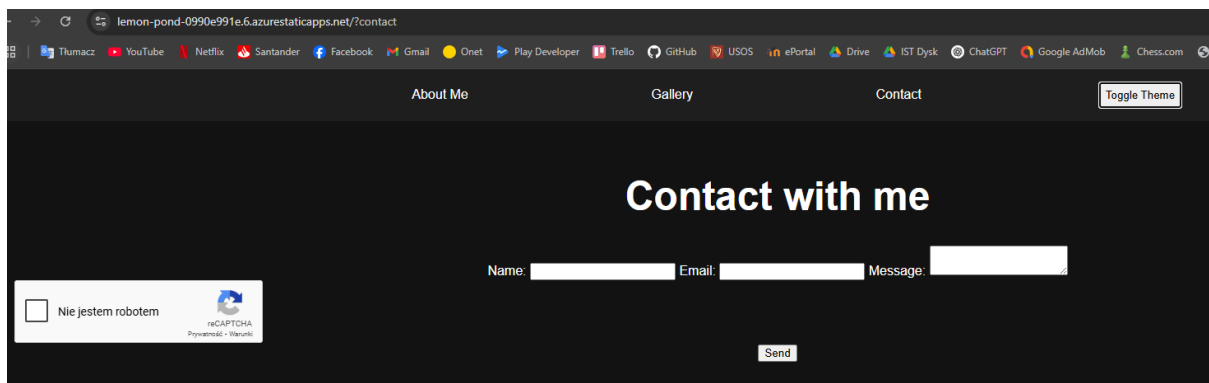
Należało przypisać domenę w systemie reCAPTCHA Google. Greaptcha renderuje widжет Google, a w form jej wysłanie jest przechwycone i sprawdzone, czy greaptcha nie zwraca błędu.

```

console.log('Contact form loaded, rendering reCAPTCHA...');

const recaptchaContainer = document.getElementById('recaptcha');
if (recaptchaContainer) {
  try {
    grecaptcha.render('recaptcha', {
      'sitekey': '6LeIxAcTAAAAAJcZVRqyHh71UMIEGNQ_MXjiZKhI'
    });
    console.log('reCAPTCHA rendered.');
```

Rysunek 6: Fragment kodu odpowiedzialnego za mechanizm reCAPTCHA



Rysunek 7: Strona kontakt w aplikacji

## 1.3 Wdrożenie aplikacji w środowisku chmurowym

### 1.3.1 Azure Static Web Apps

Azure Static Web Apps oferuje darmową usługę hostingową dla aplikacji SPA. Kroki wdrożenia:

1. Zaloguj się na <https://portal.azure.com>.
2. Utwórz nową aplikację statyczną, podając szczegóły projektu.

3. Połącz aplikację z repozytorium GitHub.
4. Wdróż aplikację i sprawdź jej działanie.

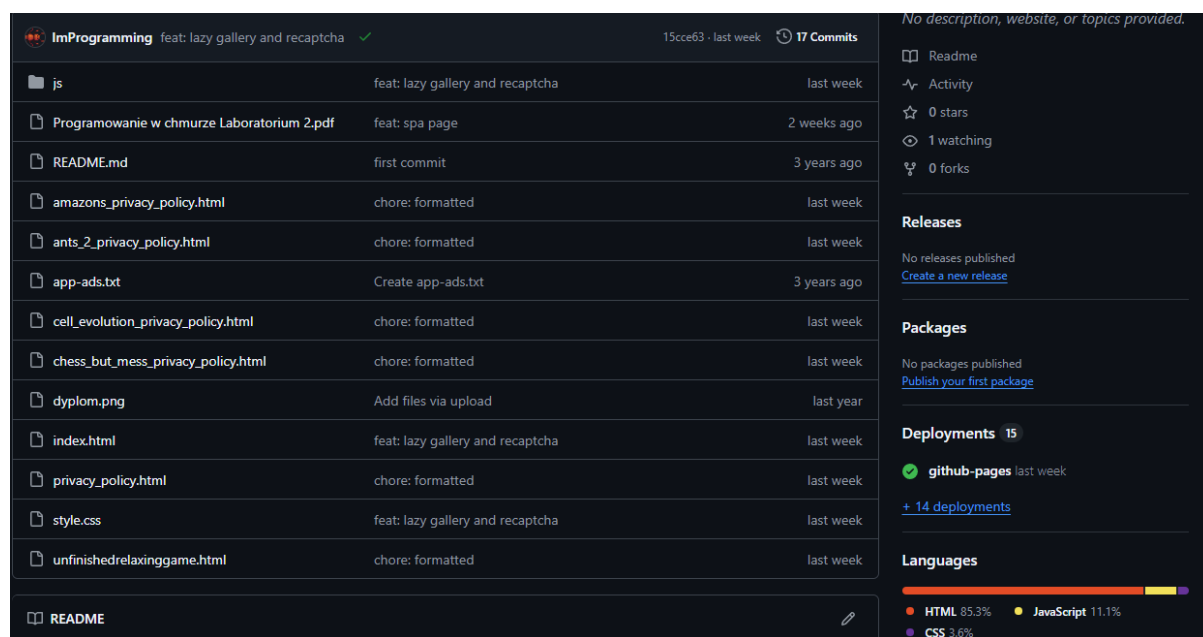
Wdrożenie aplikacji zadziałało bezproblemowo. Po zcommitowaniu kodu na GitHub i poczekaniu, aż Azure to przetworzy, zmieni się hostowana wersja aplikacji.

### 1.3.2 GitHub Pages

GitHub Pages umożliwia hostowanie aplikacji SPA. Kroki wdrożenia:

1. Utwórz repozytorium na GitHubie.
2. Skonfiguruj GitHub Pages w ustawieniach repozytorium.
3. Sprawdź dostępność aplikacji pod adresem `https://username.github.io/repository`.

Różne pliki były hostowane na `https://github.com/lmProgramming/lmProgramming.github.io` od dawna - głównie informacji dla botów reklamowych Google'a oraz polityki prywatności gier autora. Dodanie nowej struktury strony przebiegło bezproblemowo.



Rysunek 8: Udany deployment na Github pages.

## 1.4 Testowanie aplikacji

Przeprowadzone testy aplikacji:

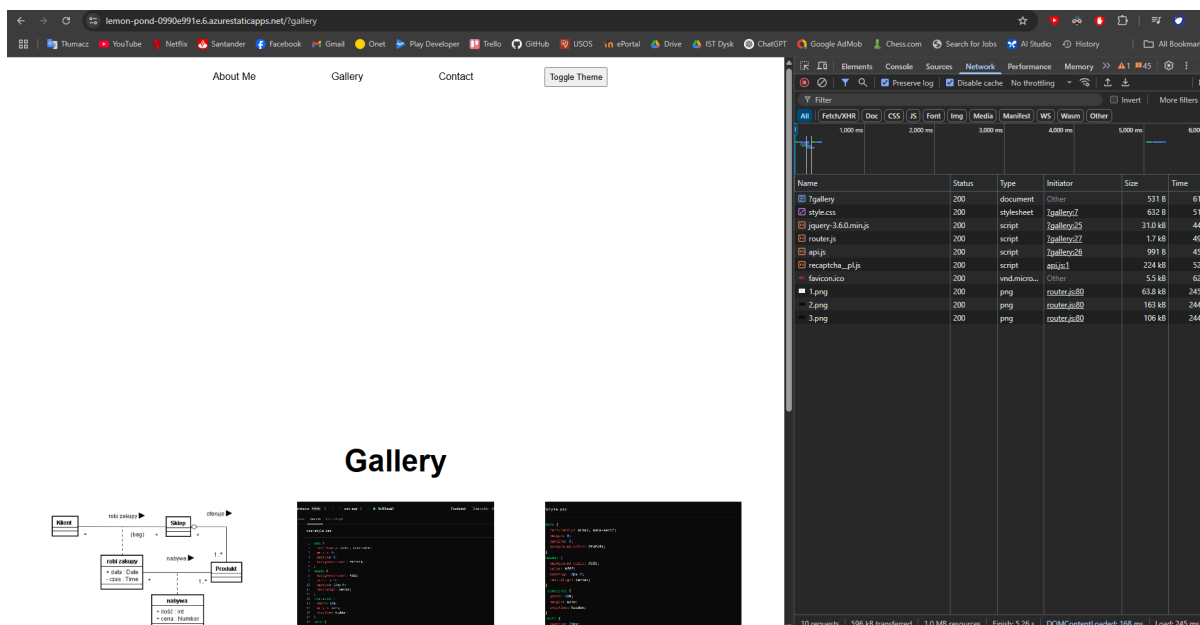
- Sprawdź nawigację między stronami.
- Zweryfikuj poprawność ładowania obrazów w galerii.
- Przetestuj walidację formularza kontaktowego.
- Użyj narzędzia Lighthouse w ChromeDevTools do analizy wydajności.



### 1.4.1 Nawigacja

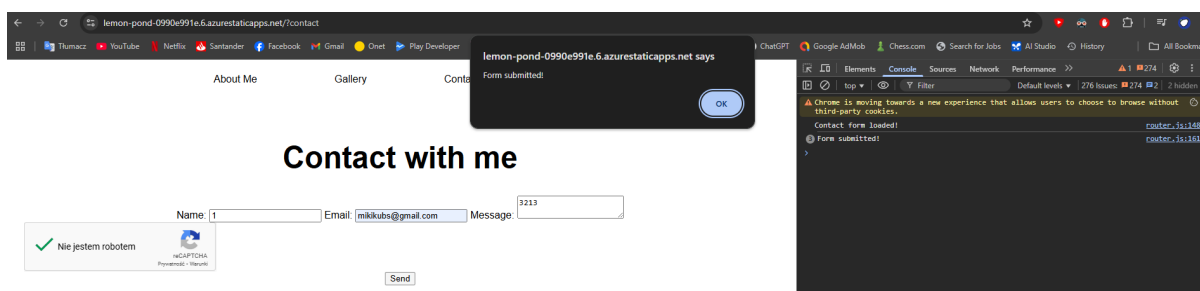
Nawigacja działa bezproblemowo, wszystkie stany są zapisane w historii. Nawigacja jest praktycznie natychmiastowa.

### 1.4.2 Leniwe ładowanie



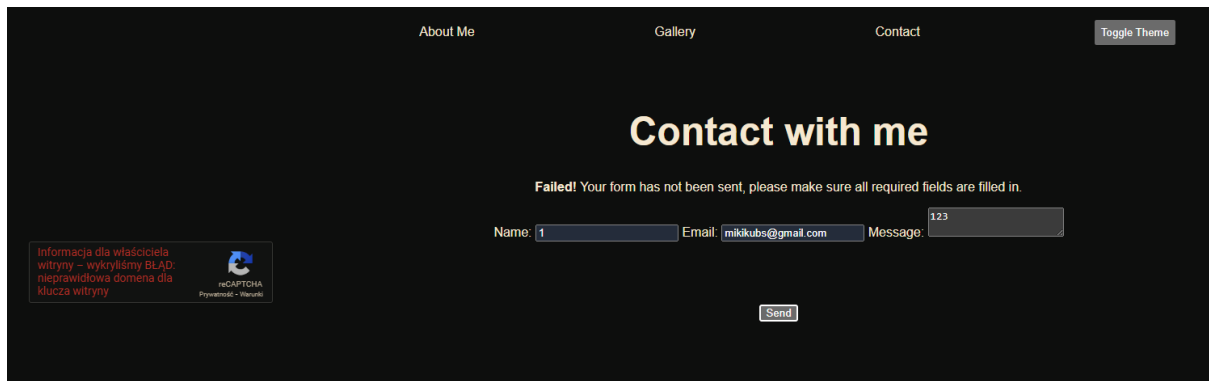
Rysunek 9: Tylko 3 z 9 obrazków załadowana, co oznacza, że leniwe ładowanie działa.

### 1.4.3 ReCAPTCHA



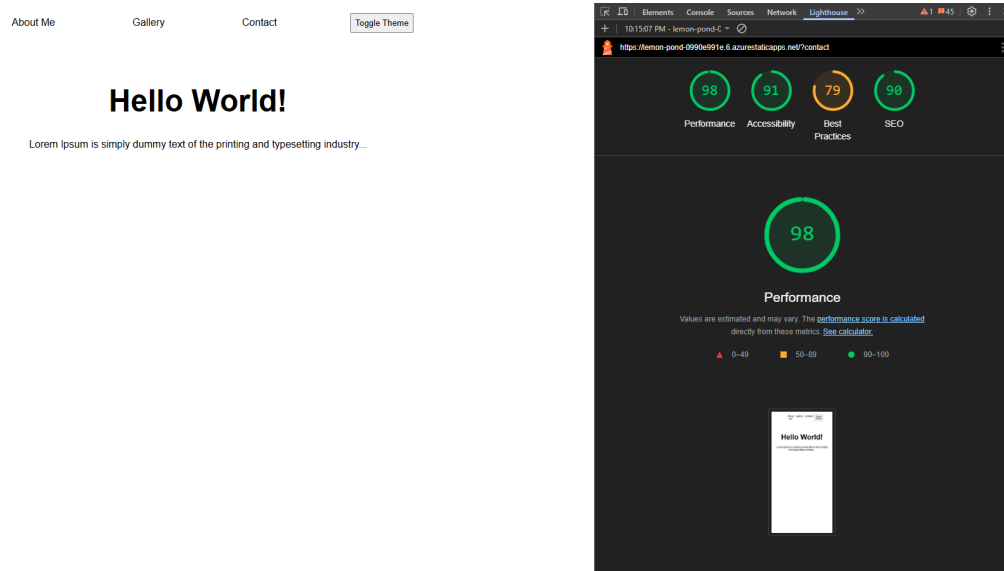
Rysunek 10: Sukces wysłania formularza z reCAPTCHA.

Przez test aplikacji na innej domenie, mechanizm reCAPTCHA nie może działać - wysłanie formularza nie uda się.



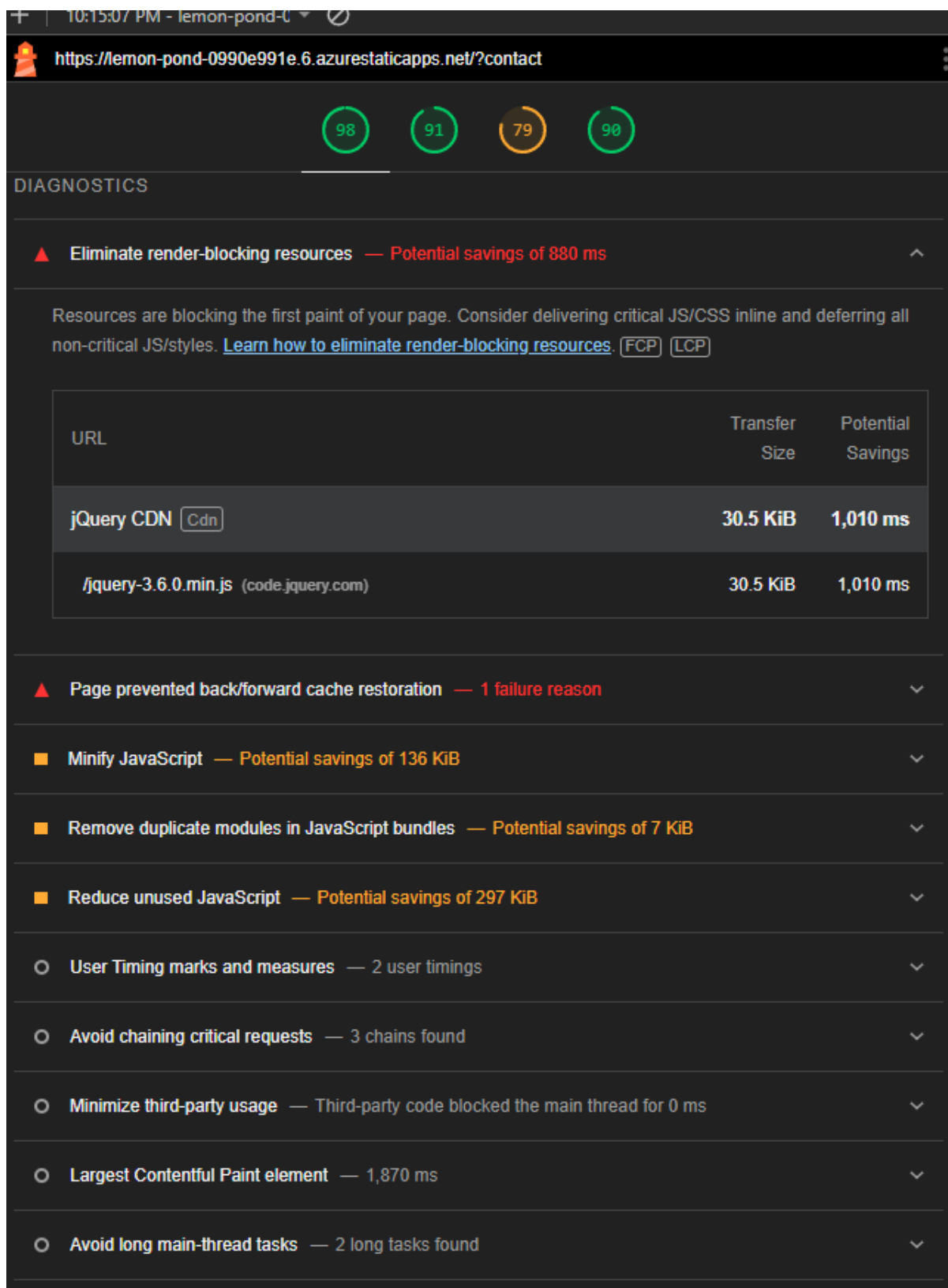
Rysunek 11: Nieudane wysłanie formularza z reCAPTCHA.

#### 1.4.4 Lighthouse



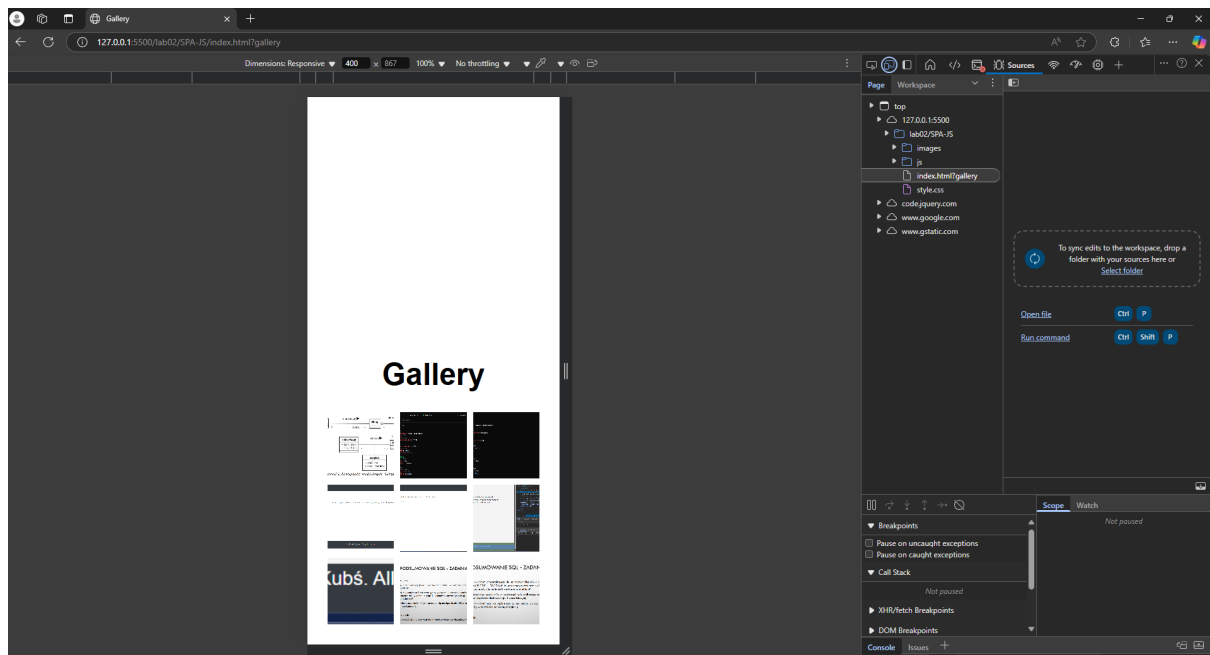
Rysunek 12: Wynik analizy Lighthouse.

Wynik analizy performance to aż 98/100, co oznacza wynik bardzo dobry, nie ma potrzeby go polepszać.



Rysunek 13: Szczegóły analizy Lighthouse.

### 1.4.5 Inna przeglądarka i rozdzielczość



Rysunek 14: Poprawne wyświetlanie strony w przeglądarce Edge, symulując wymiary telefonu