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.

2 Wykorzystane technologie

- jQuery
- Azure Static Web
- Github Pages

2.1 Tworzenie aplikacji SPA z wykorzystaniem HTML i JS

2.1.1 Kod HTML

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

```html
<!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

2.1.2 Stylizacja CSS

Plik `style.css` definiuje kaskadowe arkusze stylów:

```css
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;
}
```

2.1.3 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 />
```

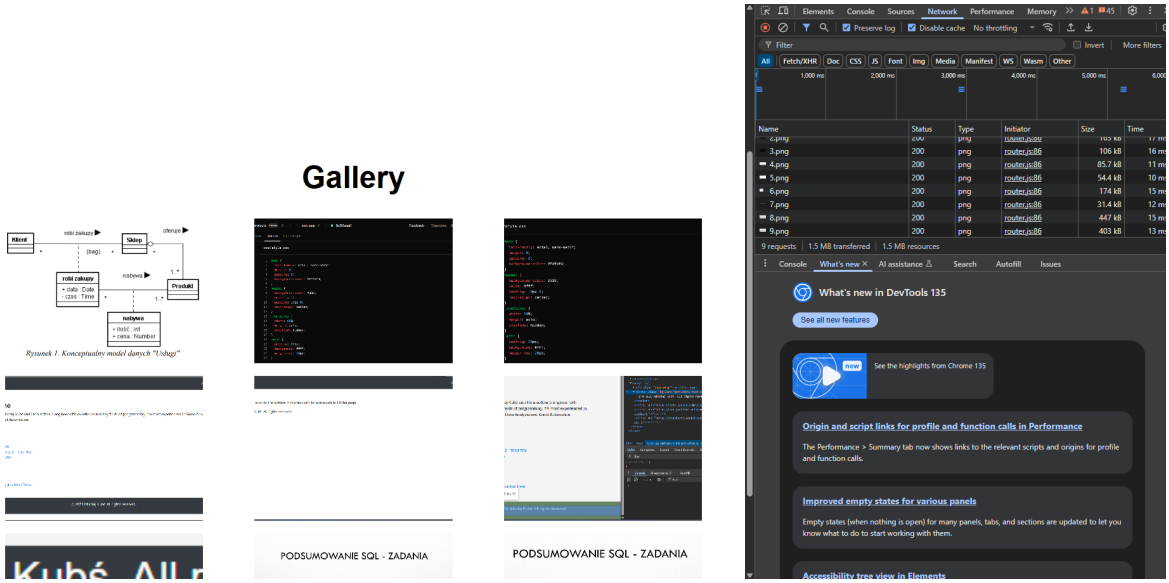
Rysunek 3: Fragment kodu router.js

2.1.4 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.

```
    <br />
    <h1 class="title">Gallery</h1>
    <div class="grid3x3" id="gallery-container"></div>
    <div id="modal" class="modal hidden">
        <span id="close-modal" class="close">&times;</span>
        <img id="modal-image" class="modal-content" />
    </div>
`;

const galleryContainer = document.getElementById('gallery-container');
const images = Array.from({ length: 9 }, (_, i) => `images/${i + 1}.png`);

images.forEach((src, index) => {
    const img = document.createElement('img');
    img.dataset.src = src;
    img.alt = `Image ${index + 1}`;
    img.classList.add('thumbnail');
    galleryContainer.appendChild(img);
});

const observer = new IntersectionObserver((entries) => {
    entries.forEach((entry) => {
        if (entry.isIntersecting) {
            const img = entry.target;
            img.src = img.dataset.src;
            img.onload = () => img.classList.add('loaded');
            observer.unobserve(img);
        }
    });
});

document.querySelectorAll('.thumbnail').forEach((img) => observer.observe(img));
```

Rysunek 4: Fragment kodu odpowiedzialnego za leniwe ładowanie



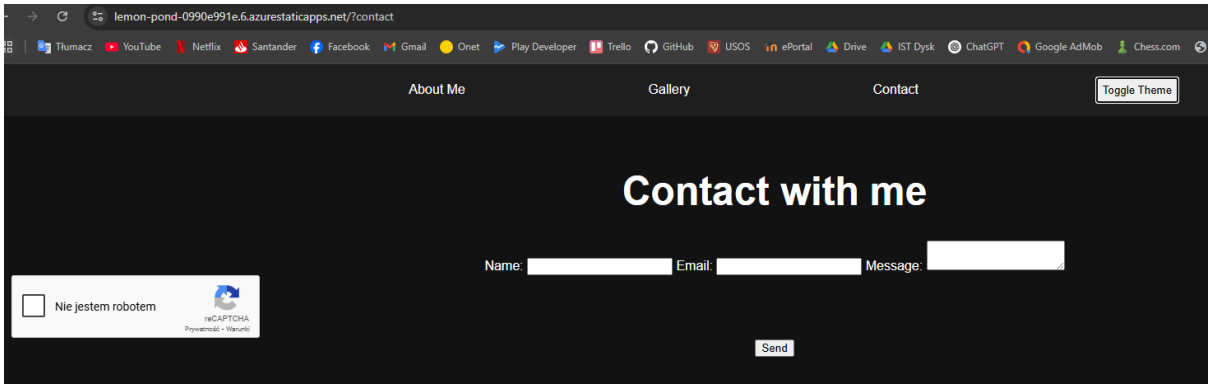
Rysunek 5: Galeria w aplikacji

2.1.5 Implementacja reCAPTCHA

Należało przypisać domenę w systemie reCAPTCHA Google. Grecaptcha renderuje widget Google, a w form jej wysłanie jest przechwycone i sprawdzone, czy grecaptcha nie zwraca błędu.

```
function RenderContactPage() {
        <label for="email">Email:</label>
        <input type="email" id="email" name="email" required />
        <label for="message">Message:</label>
        <textarea id="message" name="message" required></textarea>
        <div class="form-item">
            <div id="recaptcha"></div>
        </div>
        <button type="submit">Send</button>
    </form>
`;

    console.log('Contact form loaded!');

    grecaptcha.render('recaptcha', {
        'sitekey': '6LdcQg0rAAAAAEciRtr35TJA-q2CRoVBopEIDthL'
    });

    $(function () {
        $('#contact-form').off('submit').on('submit', function (e) {
            e.preventDefault();
            var form = $(this);

            form.find('.alert-danger').hide();

            console.log('Form submitted!');

            function showError() {
                form.find('.alert-danger').fadeIn();
            }

            if (grecaptcha.getResponse() == "") {
                showError();
                return false;
            }

            alert('Form submitted!');
        });
```

Rysunek 6: Fragment kodu odpowiedzialnego za mechanizm reCAPTCHA



Rysunek 7: Strona kontakt w aplikacji

2.2 Wdrożenie aplikacji w środowisku chmurowym

2.2.1 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.

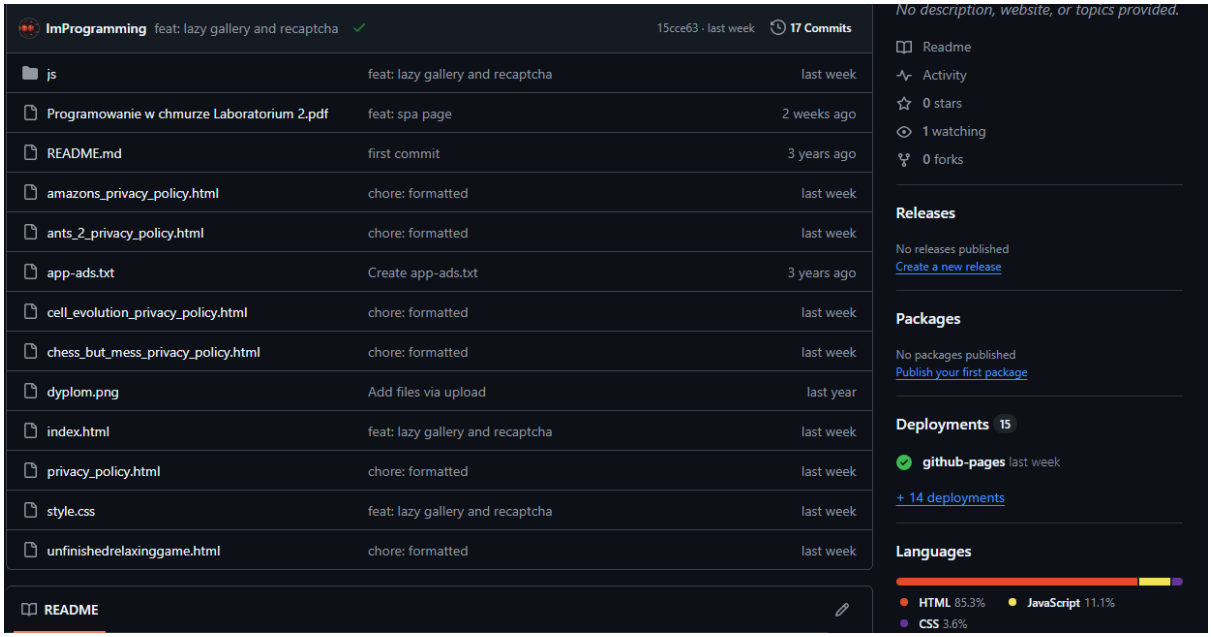
Wdrożenie aplikacji zadziałało bezproblemowo. Po zcommitowaniu kodu na GitHub i poczekaniu, aż Azure to przetworzy, zmieni się hostowana wersja aplikacji.

2.2.2 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.

2.3 Testowanie aplikacji

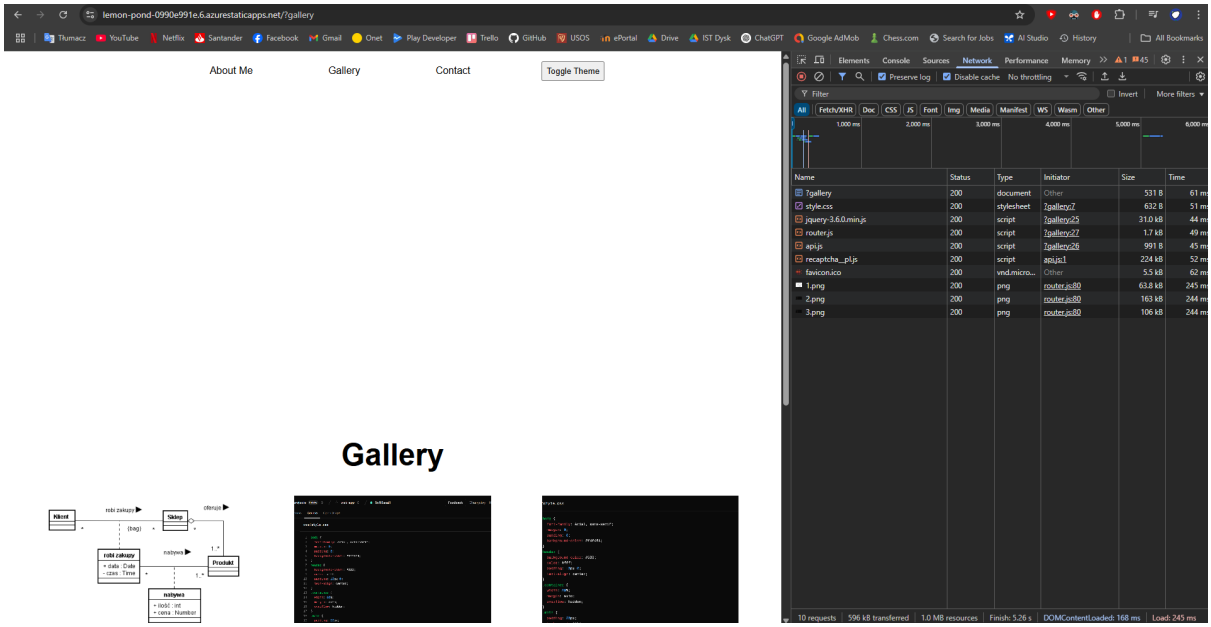
Przeprowadzone testy aplikacji:

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

2.3.1 Nawigacja

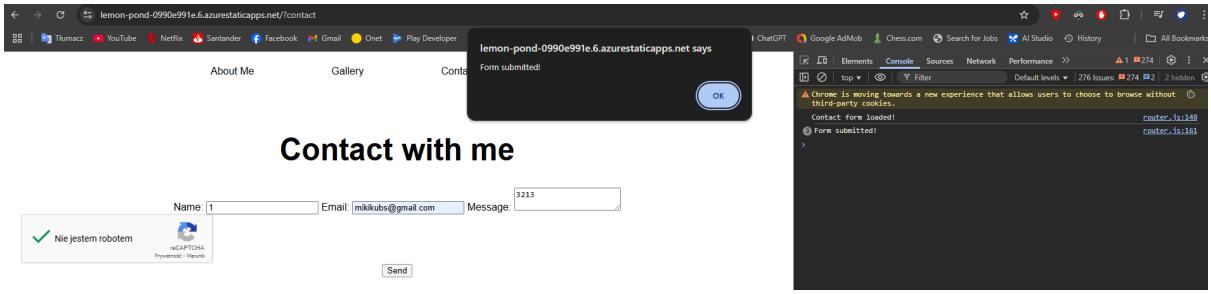
Nawigacja działa bezproblemowo, wszystkie stany są zapisane w historii. Nawigacja jest praktycznie natychmiastowa.

2.3.2 Leniwe ładowanie



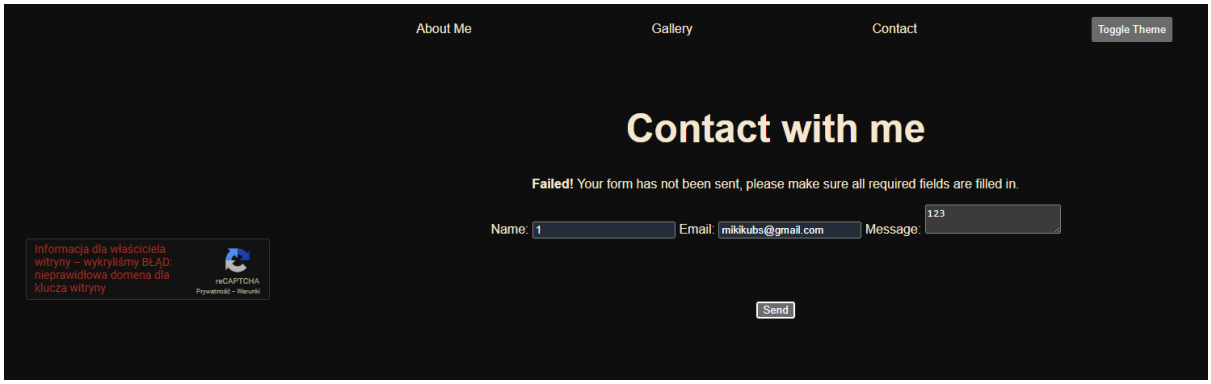
Rysunek 9: Tylko 3 z 9 obrazków załadowana, co oznacza, że leniwe ładowanie działa.

2.3.3 ReCAPTCHA



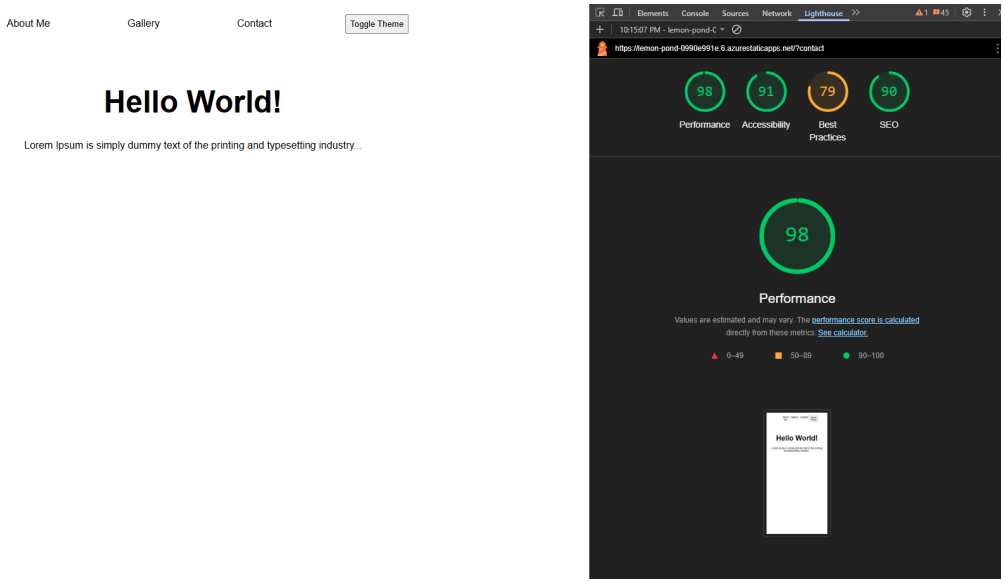
Rysunek 10: Sukces wysłania formularza z reCAPTCHA.

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



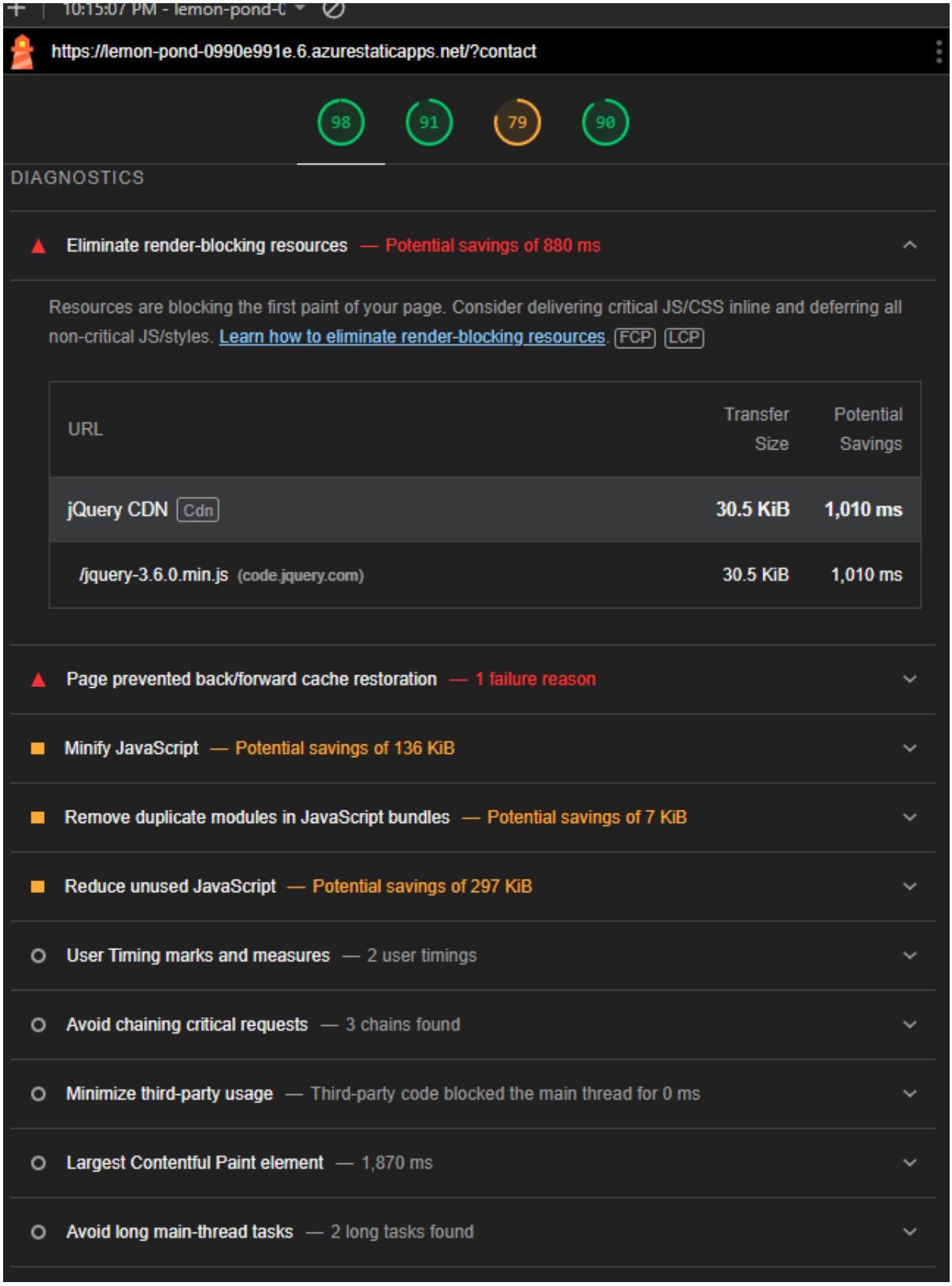
Rysunek 11: Nieudane wysłanie formularza z reCAPTCHA.

2.3.4 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.