|  |  |
| --- | --- |
| **Politechnika Świętokrzyska** | |
| Wydział Elektrotechniki, Automatyki i Informatyki | |
| Autor:  Przemysław Kałuziński  Michał Kaczor  Jakub Kuśmierczyk | Grupa dziekańska:  1IZ22B |
| Numery indeksów:  091271 | Data wykonania: 03.01.2025 |
| **Menedżer i analizator haseł** | |

[**1.** **Opis aplikacji**](#_Toc2133791100)

[**2.** **Kluczowe funkcje**](#_Toc272056189)

[**3.** **Zastosowane technologie**](#_Toc519575795)

[**4.** **Wygląd aplikacji i sposób użycia aplikacji**](#_Toc747179536)

[**5.** **Najważniejsze fragmenty kodu**](#_Toc1964396124)

# Opis aplikacji

Aplikacja "Menedżer i Analizator Haseł" to narzędzie, które umożliwia użytkownikom bezpieczne przechowywanie własnych haseł oraz analizowanie ich siły. Jest przeznaczona dla osób ceniących bezpieczeństwo w świecie cyfrowym, pomagając im zarządzać poufnymi danymi w jednym miejscu.

# Kluczowe funkcje

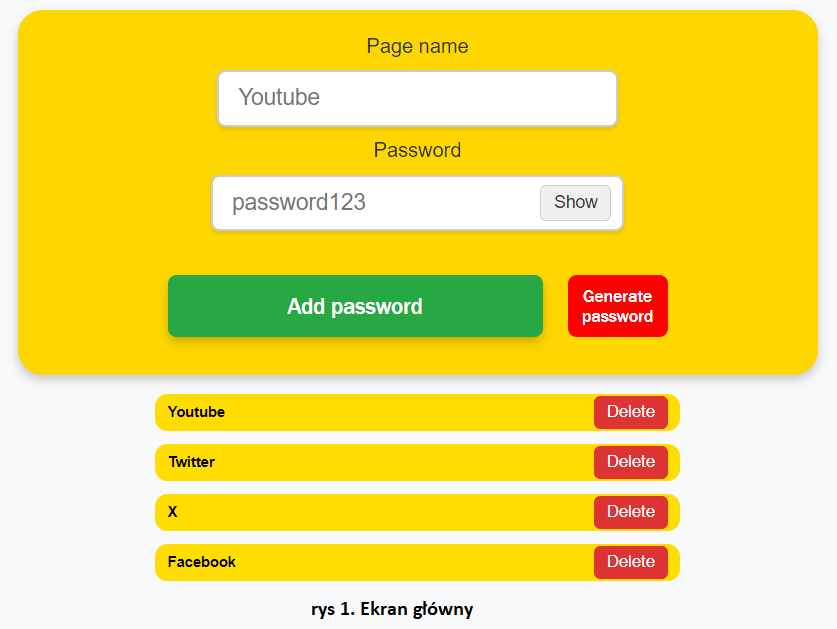
1. **Bezpieczne Przechowywanie Haseł**
   1. Użytkownicy mogą dodawać swoje hasła do bazy danych aplikacji.
   2. Wszystkie przechowywane hasła są szyfrowane za pomocą zaawansowanych algorytmów, takich jak AES-256, zapewniając najwyższy poziom ochrony.
2. **Analiza Siły Haseł**
   1. Aplikacja ocenia siłę każdego dodanego hasła w czasie rzeczywistym.
   2. System bierze pod uwagę takie aspekty jak długość, użycie małych i wielkich liter, cyfr oraz znaków specjalnych.
   3. Dostarcza szczegółowe wskazówki dotyczące poprawy słabych haseł.
3. **Generator Silnych Haseł**
   1. Umożliwia generowanie losowych, silnych haseł
4. **Synchronizacja i Dostępność**
   1. Obsługa wielu urządzeń za pomocą synchronizacji w chmurze z zachowaniem pełnej prywatności użytkownika.
5. **Intuicyjny Interfejs Użytkownika**
   1. Prosta i przyjazna nawigacja dla użytkowników o każdym poziomie zaawansowania.

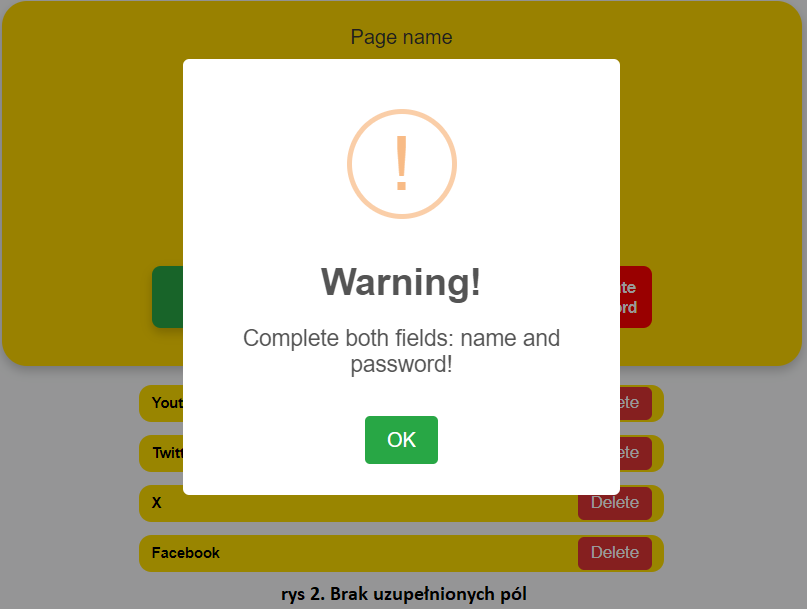
# Zastosowane technologie

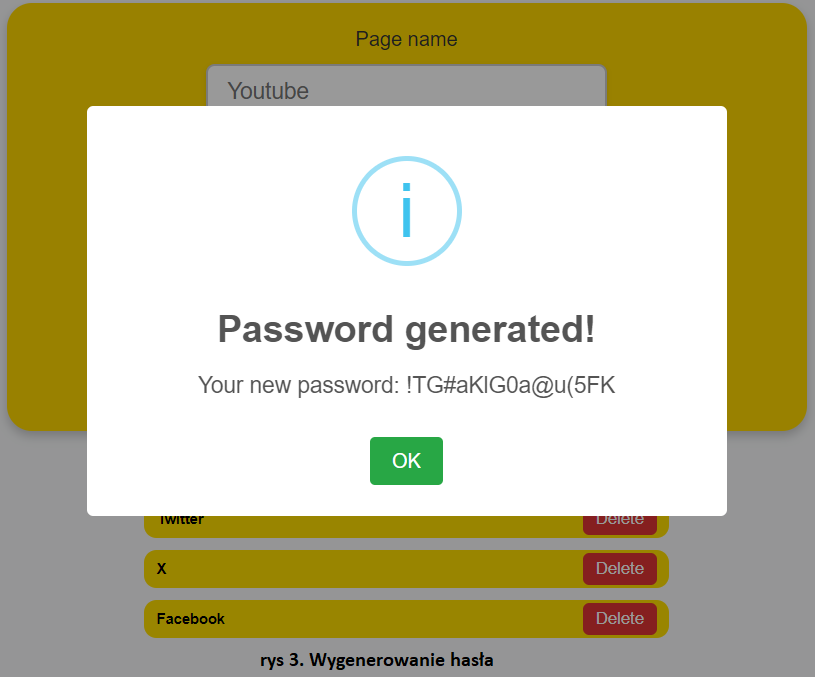
Aplikacja opiera się na nowoczesnym stosie technologicznym, który zapewnia wysoką wydajność, skalowalność i bezpieczeństwo.

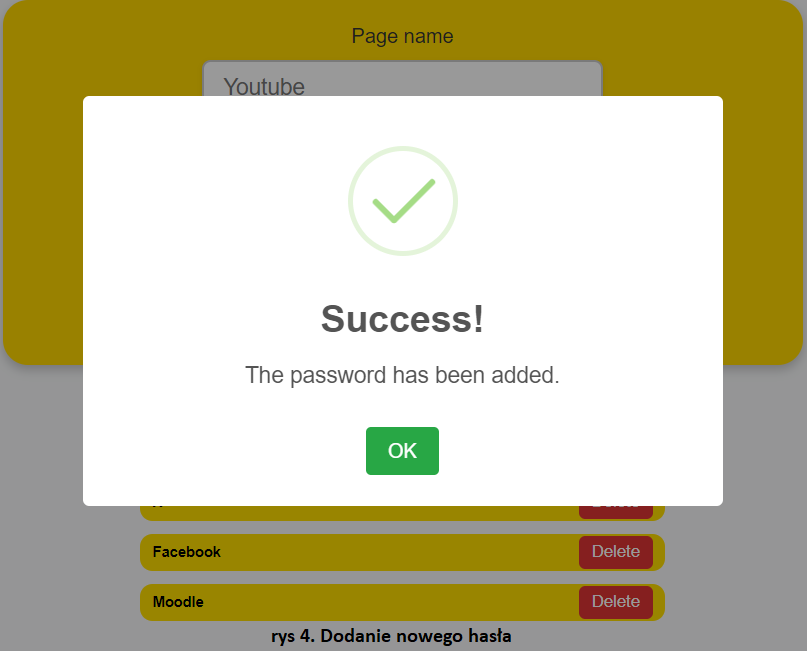
* **Vite i React**: Umożliwiają szybkie prototypowanie oraz wydajną obsługę nowoczesnych aplikacji SPA (Single Page Application).
* **JavaScript i Express.js**: Zapewniają prostą integrację z bibliotekami takimi jak zxcvbn oraz elastyczność w tworzeniu rozwiązań serwerowych.
* **MySQL**: Dzięki swojej niezawodności i popularności idealnie nadaje się do zarządzania relacyjnymi danymi aplikacji.
* **zxcvbn**: Pozwala na szczegółową i zaawansowaną analizę haseł, co jest kluczowe dla głównej funkcjonalności aplikacji.

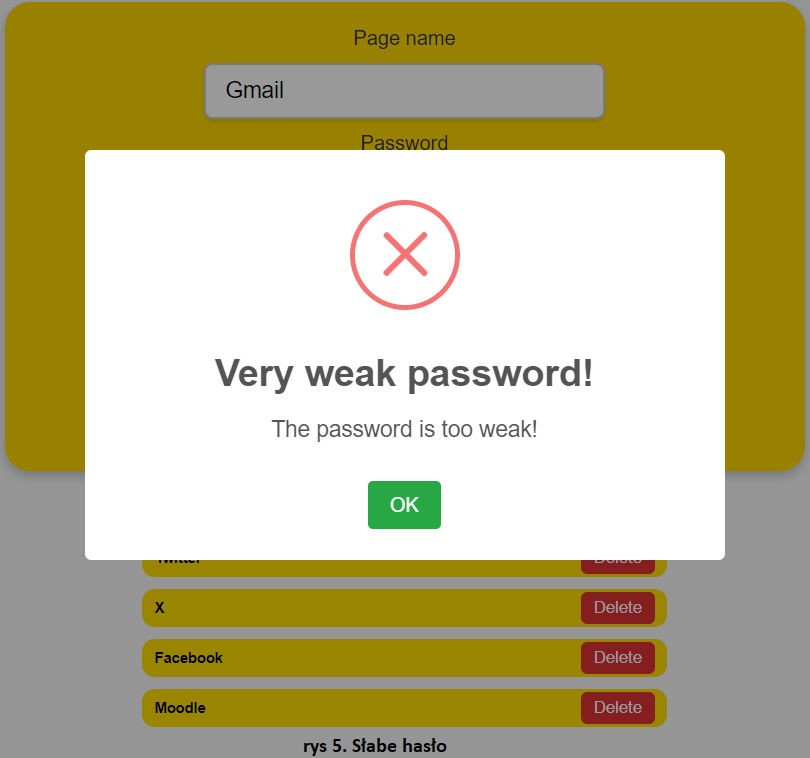
# Wygląd aplikacji i sposób użycia aplikacji

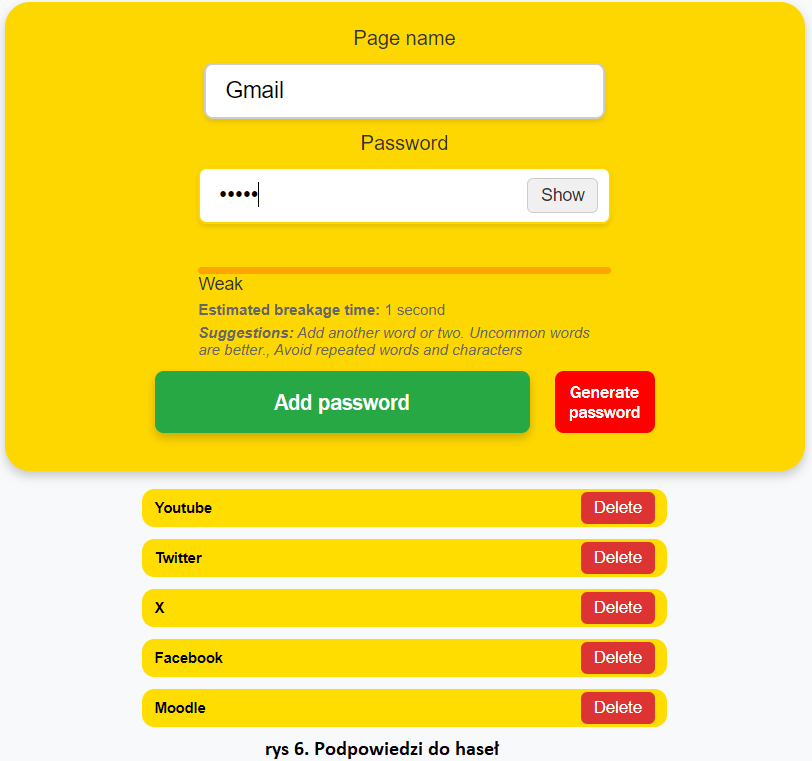
Aplikacja (rys 1.) posiada dwa pola, które należy uzupełnić aby dodać nowe hasło do menedżera.

Jeśli pola nie zostaną uzupełnione i naciśniemy przycisk “Add password” wyświetli się komunikat ostrzegawczy (rys 2.). 

Jeśli użytkownik będzie chciał wygenerować nowe silne hasło może wybrać przycisk “Generate password”, po czym otrzyma odpowiedni komunikat (rys 3.). 

Po wypełnieniu obu pól i wybraniu przycisku “Add password” wyświetli się odpowiedni komunikat (rys 4.). 

Natomiast w przypadku wpisania zbyt słabego hasła otrzymamy komunikat, iż podane hasło jest zbyt słabe (rys 5.). 

Gdy użytkownik wpisuje hasło pod polem hasła wyświetlają się pasek siły hasła, czas potrzebny na złamanie hasła oraz podpowiedzi jak można poprawić hasło (rys 6.). 

Wpisywane hasło można również podejrzeć wybierając przycisk “Show” lub ukryć przyciskiem “Hide”. Wcześniej zapisane hasła można podejrzeć poprzez naciśnięcie na nie w tabeli po czy można je ponownie ukryć ponownym kliknięciem. Dodane hasła można usunąć z zapisanych.

# Najważniejsze fragmenty kodu

|  |
| --- |
| // Dodawanie hasła  app.post("/addpassword", async (req, res) => {  const { password, title } = req.body;  // Sprawdzanie siły hasła  const passwordStrength = zxcvbn(password);  const score = passwordStrength.score; // 0-4, gdzie 4 to najbezpieczniejsze  if (score < 1) {  const warningMessage = "Hasło jest zbyt słabe.";  return res.status(400).send({ message: warningMessage });  }  const hashedPassword = encrypt(password);  db.query(  "INSERT INTO passwords (password, title, iv) VALUES(?, ?, ?)",  [hashedPassword.password, title, hashedPassword.iv],  (err, result) => {  if (err) {  console.log(err);  res.status(500).send("Błąd serwera podczas dodawania hasła.");  } else {  res.send({ id: result.insertId, iv: hashedPassword.iv });  }  }  );  }); |
| const encrypt = (password) => {  const iv = Buffer.from(crypto.randomBytes(16));  const cipher = crypto.createCipheriv(  "aes-256-ctr",  Buffer.from(secret),  iv  );  const encryptedPassword = Buffer.concat([  cipher.update(password),  cipher.final()  ]);  return {  iv: iv.toString("hex"),  password: encryptedPassword.toString("hex")  };  };  const decrypt = (encryption) => {  const decipher = crypto.createDecipheriv(  "aes-256-ctr",  Buffer.from(secret),  Buffer.from(encryption.iv, "hex")  );  const decryptedPassword = Buffer.concat([  decipher.update(Buffer.from(encryption.password, "hex")),  decipher.final()  ]);  return decryptedPassword.toString();  }; |