



POLITECHNIKA RZESZOWSKA

im. Ignacego Łukasiewicza

WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

Michał Koryl, Filip Kosiorowski, Katarzyna Kośniowska

173160,173161,173162

Bezpieczeństwo i ochrona danych

Projekt zaliczeniowy

Rzeszów 2025

Spis treści

1. Wstęp	3
2. Użyte struktury	4
Spring Security	4
Google reCAPTCHA v2	7
3. Szyfrowanie haseł.....	9
4. Walidacja danych przy rejestracji użytkownika.....	11
Sprawdzanie dostępności loginu	11
Konieczność dwukrotnego wprowadzenia hasła	12
Minimalna długość hasła	12
Blokada niedozwolonych znaków	13
5. Walidacja pól wprowadzania danych.....	14
Walidacja po stronie klienta	14
Walidacja po stronie serwera	15
6. Zabezpieczenie przed atakami CSRF	16
7. Podsumowanie	17

1. WSTEP

Celem projektu było stworzenie prostego sklepu internetowego, uwzględniającego kluczowe mechanizmy ochrony danych i zwiększania bezpieczeństwa użytkowników. W trakcie pracy szczególną uwagę poświęcono implementacji rozwiązań, które zabezpieczają aplikację przed popularnymi zagrożeniami oraz poprawiają jej niezawodność w kontekście ochrony danych wrażliwych.

Dokładny opis sklepu znajduje się w sprawozdaniu na przedmiot Aplikacje Internetowe, w tym skupimy się głównie na części zabezpieczeń.

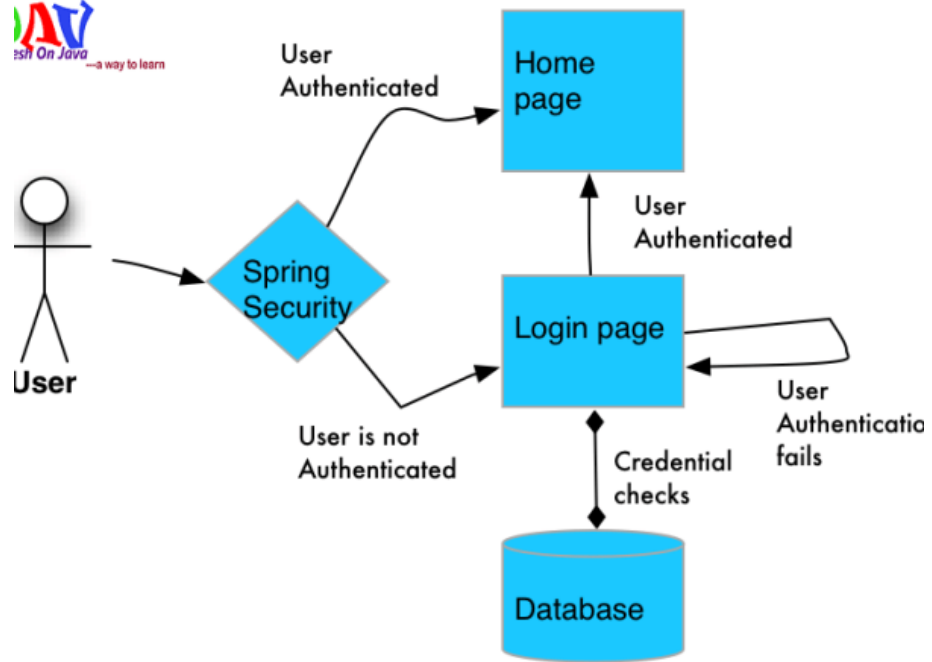
2. UŻYTE STRUKTURY

SPRING SECURITY

Framework wykorzystano do implementacji mechanizmu logowania oraz zarządzania bezpieczeństwem aplikacji. Spring Security jest zaawansowanym modułem w ekosystemie Spring, który dostarcza gotowe rozwiązania pozwalające chronić aplikacje przed najczęstszymi zagrożeniami, takimi jak nieautoryzowany dostęp czy ataki typu brute force.

Kluczowe elementy implementacji:

- 1) **Uwierzytelnianie użytkowników:** Spring Security został skonfigurowany do obsługi logowania za pomocą formularza. Dane logowania użytkowników, w tym hasła, są przechowywane w bazie danych w postaci zaszyfrowanej, wykorzystując algorytm *bcrypt*. Dzięki temu nawet w przypadku wycieku danych hasła pozostają nieczytelne dla potencjalnych atakujących.
- 2) **Autoryzacja dostępu:** Framework umożliwia kontrolowanie dostępu do zasobów aplikacji na podstawie ról użytkowników. Dzięki temu poszczególne sekcje sklepu, np. panel administracyjny, są dostępne wyłącznie dla uprawnionych osób.
- 3) **Obsługa błędnych logowań:** Spring Security umożliwia łatwą konfigurację zabezpieczeń przed wielokrotnymi błędnymi próbami logowania, co w połączeniu z reCAPTCHA stanowi dodatkową warstwę ochrony przed atakami siłowymi (*brute force*).
- 4) **Filtry bezpieczeństwa:** Wdrożono niestandardowy filtr walidujący dane wprowadzane przez użytkownika, aby uniemożliwić wstrzykiwanie niebezpiecznych treści, takich jak złośliwy kod SQL lub JavaScript.



ZALOGUJ SIĘ

Email •

Hasło •



ZALOGUJ SIĘ

[Zapomniales hasła?](#)

[Nie masz konta? Zarejestruj się tutaj](#)

REJESTRACJA

Login •

Hasło •



Powtórz hasło •



DOŁĄCZ

[Masz już konto? Zaloguj się tutaj!](#)

GOOGLE RECAPTCHA V2

W celu zwiększenia bezpieczeństwa logowania w aplikacji, zaimplementowano mechanizm recaptcha, który uruchamia się po trzech nieudanych próbach zalogowania. Mechanizm ten stanowi dodatkową warstwę ochrony, skutecznie zapobiegając atakom zautomatyzowanych botów oraz brute force.

Opis działania i integracji:

1) **Warunkowe wyświetlanie reCAPTCHA:**

Mechanizm został skonfigurowany tak, aby aktywował się wyłącznie po trzech nieudanych próbach logowania. Oznacza to, że użytkownik wprowadzający błędne dane logowania kilka razy z rzędu musi dodatkowo potwierdzić swoją tożsamość, rozwiązując reCAPTCHA. Pozwala to na wygodne logowanie dla większości użytkowników i jednocześnie utrudnia działania potencjalnych atakujących.

2) **Integracja z formularzem logowania:**

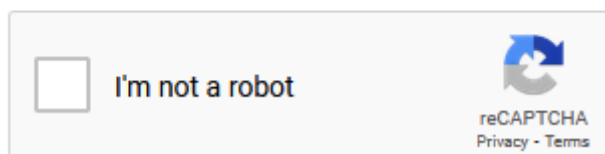
W momencie, gdy liczba nieudanych prób logowania przekroczy określony próg, reCAPTCHA jest dynamicznie wyświetlana na ekranie logowania. Formularz wysyła dane do serwera, który weryfikuje poprawność tokenu za pomocą API Google.

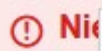
3) **Weryfikacja po stronie serwera:**

Po przesłaniu tokenu serwer komunikuje się z API Google, aby sprawdzić, czy użytkownik przeszedł weryfikację. W przypadku pozytywnej weryfikacji użytkownik może kontynuować proces logowania. Jeśli jest niepoprawny lub nie został przesłany, serwer odrzuca żądanie.

ZALOGUJ SIE

[Zapomniales hasla?](#)





Sp
pop

Email •

Email jes

Hasło •



Nie

Select all images with
a fire hydrant



VERIFY



I'm not a robot



reCAPTCHA
Privacy - Terms

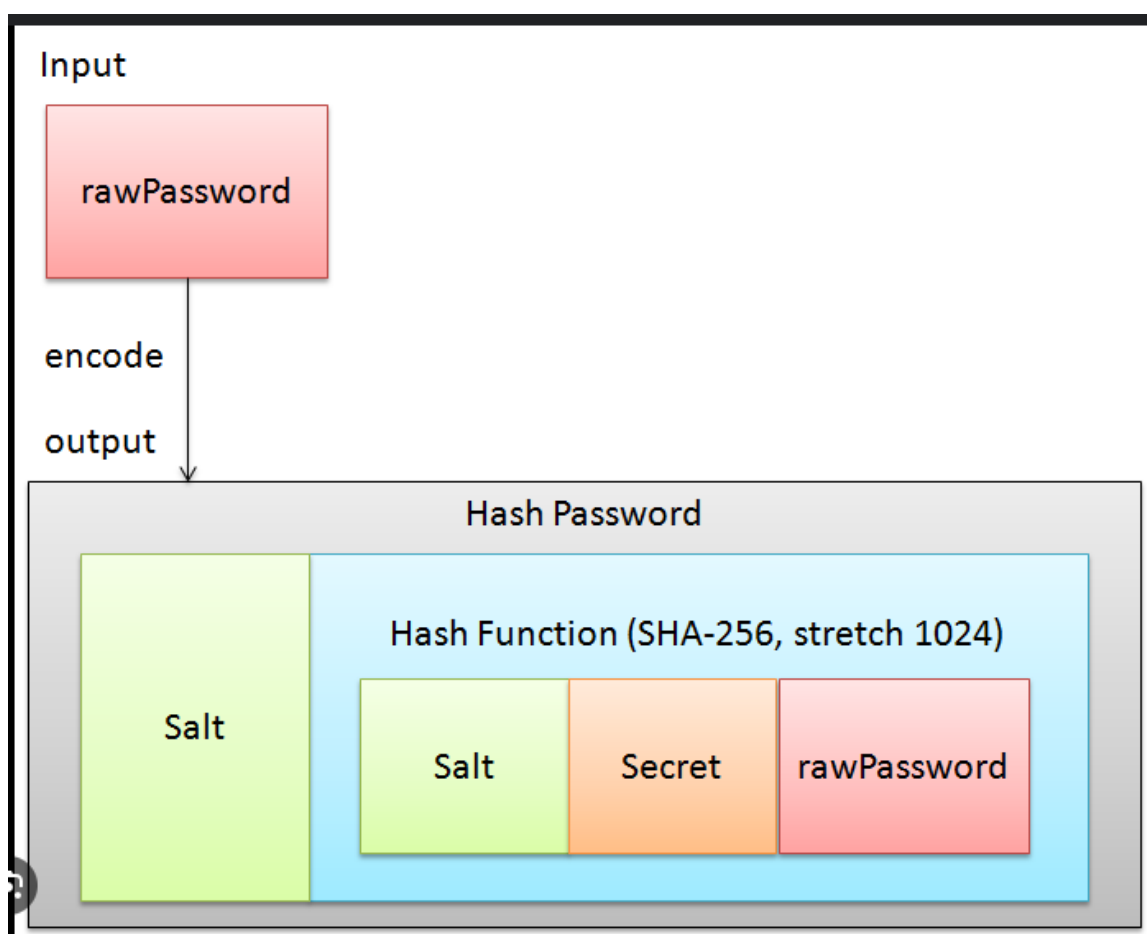
Nie masz konta? Zarejestruj się tutaj

3. SZYFROWANIE HASEŁ

W celu zapewnienia bezpieczeństwa danych logowania użytkowników, w projekcie wykorzystano mechanizm szyfrowania haseł przy użyciu klasy BCryptPasswordEncoder, dostarczanej przez Spring Security. Algorytm jest jednym z najbardziej rekomendowanych algorytmów do przechowywania haseł, dzięki swojej odporności na ataki siłowe i zdolności do generowania unikalnych hashy nawet dla identycznych haseł.

BCrypt to algorytm haszujący, który:

- 1) Wykorzystuje losowy salt (wartość losową dodawaną do hasła przed jego haszowaniem), co uniemożliwia tworzenie tęczyowych tablic (*rainbow tables*).
- 2) Jest oparty na adaptacyjnej funkcji skrótu (*adaptive hashing function*), co pozwala zwiększyć złożoność operacji wraz ze wzrostem mocy obliczeniowej komputerów.
- 3) Generuje unikalny hash dla każdego hasła, nawet jeśli dane wejściowe są identyczne.



Szyfrowanie haseł przy rejestracji:

```
public boolean registerUser(User user, String roleName){
    if (repository.existsByUsername(user.getUsername())) {
        return false; // Użytkownik o podanym loginie już istnieje
    }

    Role role = roleRepository.findByName(roleName);
    if (role == null) {
        throw new RuntimeException("Rola nie istnieje");
    }

    user.setPassword(passwordEncoder.encode(user.getPassword()));

    Cart cart = new Cart();
    cart.setUser(user); // Powiązanie koszyka z użytkownikiem

    repository.save(user);

    UserRole userRole = new UserRole();
    userRole.setUser(user); // Przypisanie użytkownika
    userRole.setRole(role); // Przypisanie roli
    userRoleRepository.save(userRole);

    return true;
}
```

Wygląd bazy danych:

	ID	PASSWORD	USERNAME
1	1	\$2a\$12\$6xpZ81nwiMDAbWe4xM/Vs.QuunRPSbhfUBE2X9wIFejAd...	admin
2	21	\$2a\$10\$PkmzucJH/jXL0ishtAU.h.jm4HPEwL3L4odk0ifw9oPrm...	TymbarkPomarancza
3	22	\$2a\$10\$32i0WChmLSFSeWI7C31L4erJpdXySAjNM2WAPTEIDiipH...	KasiaMasia
4	23	\$2a\$10\$SJBS2sDunin0Iby7A4eRXeQjL.RnNEIgpsNec5w6LFZtZ...	user09

4. WALIDACJA DANYCH PRZY REJESTRACJI UŻYTKOWNIKA

SPRAWDZANIE DOSTĘPNOŚCI LOGINU

Podczas rejestracji weryfikowane jest, czy podany login (nazwa użytkownika) jest już zarejestrowany w systemie. Mechanizm ten pozwala uniknąć duplikacji kont i gwarantuje unikalność identyfikatora użytkownika w bazie danych.

W przypadku, gdy login już istnieje, użytkownik otrzymuje odpowiedni komunikat na ekranie rejestracji.

REJESTRACJA

Login •

TymbarkPomarańcza

Użytkownik o tej nazwie już istnieje!

Hasło •

••••••••



Powtórz hasło •

••••••••



DOŁĄCZ

Masz już konto? Zaloguj się tutaj!

KONIECZNOŚĆ DWUKROTNEGO WPROWADZENIA HASŁA

Formularz rejestracyjny wymaga od użytkownika dwukrotnego wprowadzenia hasła. Mechanizm ten służy do eliminacji błędów spowodowanych przypadkowymi literówkami. Hasła są porównywane przed zapisaniem do bazy danych.

Jeśli wprowadzone hasła się różnią, proces rejestracji zostaje przerwany, a użytkownik otrzymuje stosowny komunikat.

Hasło •

••••••••

👁

Hasła się różnią

Powtórz hasło •

••••••••

👁

DOŁĄCZ

MINIMALNA DŁUGOŚĆ HASŁA

Wprowadzono wymóg, aby hasło miało co najmniej 8 znaków. Zapewnia to minimalny poziom trudności hasła, utrudniając jego odgadnięcie.

Hasło •

•

👁

Hasło powinno mieć minimum 8 znaków

Powtórz hasło •

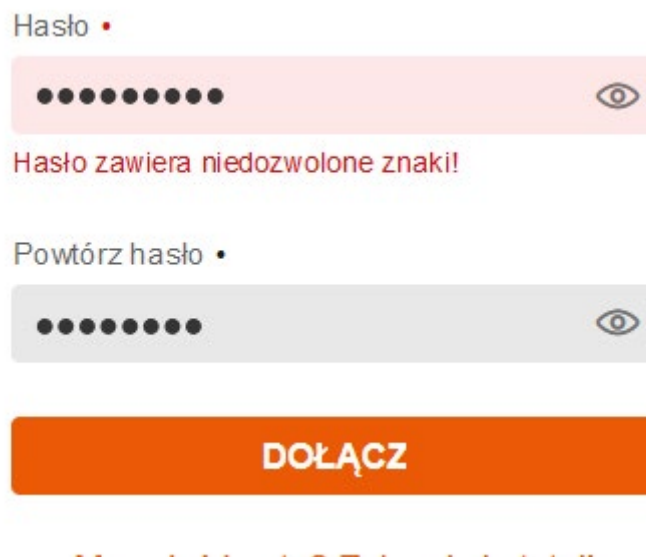
••••••••

👁

DOŁĄCZ

BLOKADA NIEDOZWOLONYCH ZNAKÓW

Hasło nie może zawierać znaków specjalnych, takich jak ;, /, <, > itp., które mogłyby zostać użyte do prób ataków, takich jak SQL Injection czy Cross-Site Scripting (XSS).



The image shows a registration form with two password input fields. The first field, labeled 'Hasło', is highlighted in red and contains ten black dots. To its right is a red eye icon. Below this field is a red error message: 'Hasło zawiera niedozwolone znaki!'. The second field, labeled 'Powtórz hasło', is highlighted in grey and also contains ten black dots with a grey eye icon to its right. Below the second field is an orange button with the text 'DOŁĄCZ'. At the bottom of the form, there is a row of small, faint orange dots.

Mechanizmy walidacji przy rejestracji użytkownika skutecznie zabezpieczają aplikację przed nieprawidłowymi danymi oraz potencjalnymi atakami. Unikalność loginu zapobiega powielaniu kont użytkowników. Dwukrotne wprowadzanie hasła zmniejsza ryzyko błędów po stronie użytkownika. Ograniczenia dotyczące długości hasła oraz niedozwolonych znaków podnoszą poziom bezpieczeństwa i ochrony przed wstrzyknięciem złośliwego kodu.

5. WALIDACJA PÓŁ WPROWADZANIA DANYCH

W projekcie zadbano o odpowiednią walidację danych wprowadzanych przez użytkowników w formularzach, w tym w polu dla numeru telefonu. Mechanizm ten uniemożliwia wpisywanie nieprawidłowych wartości, takich jak litery, znaki specjalne czy inne niedozwolone znaki, które mogłyby spowodować błędy w działaniu aplikacji lub wprowadzić złośliwy kod.

WALIDACJA PO STRONIE KLIENTA

W polach takich jak numer telefonu zastosowano ograniczenia za pomocą atrybutów HTML oraz wyrażeń regularnych.

Pole wprowadzenia numeru telefonu, posiadające ograniczenia wejścia tylko 9 cyfr

```
<input
slot="input"
type="text"
id="input-vaadin-text-field-50"
required="" maxlength="9" minlength="9" pattern="\d{9}"
aria-labelledby="label-vaadin-text-field-34">
```

Pole wprowadzenia kodu pocztowego, posiadające ograniczenie wejścia „XX-XXX”

```
<input slot="input"
type="text"
id="input-vaadin-text-field-48"
required="" pattern="\d{2}-\d{3}"
aria-labelledby="label-vaadin-text-field-28">
```

- 1) **Atrybut pattern** – wymusza, aby wprowadzone dane pasowały do wyrażenia regularnego, które pozwala na wpisanie jedynie cyfr o długości od 9 do 15 znaków.
- 2) **Atrybut title** – wyświetla wskazówkę użytkownikowi w przypadku błędnego wprowadzenia danych.
- 3) **Atrybut required** – wymaga wprowadzenia wartości w polu.
- 4) **Atrybut maxlength** oraz **minlength** – wymaga wprowadzenia minimum i maksimum liczb

WALIDACJA PO STRONIE SERWERA

Dodatkowo, dla zwiększenia bezpieczeństwa, dane wprowadzone przez użytkownika są weryfikowane także po stronie serwera. Przykład walidacji pola numeru telefonu w kodzie Java:

```
public void validatePhoneNumber(String phoneNumber) throws Exception {  
    if (!phoneNumber.matches(regex: "[0-9]{9,9}$")) {  
        throw new Exception("Numer telefonu jest nieprawidłowy. Dozwolone są tylko cyfry (9 znaków).");  
    }  
}
```

- 1) Metoda `matches()` sprawdza, czy numer telefonu składa się wyłącznie z cyfr i czy jego długość mieści się w określonym przedziale.
- 2) W przypadku nieprawidłowych danych proces rejestracji lub zapis formularza jest przerywany, a użytkownik otrzymuje stosowny komunikat.

Walidacja pól takich jak numer telefonu została zaimplementowana zarówno po stronie klienta (frontend), jak i po stronie serwera (backend). Walidacja po stronie klienta zapewnia lepszą wygodę użytkownika, zapobiegając wysyłaniu błędnych danych. Walidacja po stronie serwera gwarantuje bezpieczeństwo aplikacji, chroniąc przed manipulacjami i złośliwym kodem. Dzięki temu mechanizmowi użytkownicy mogą wprowadzać dane zgodne z założeniami, co zwiększa stabilność i bezpieczeństwo systemu.

6. ZABEZPIECZENIE PRZED ATAKAMI CSRF

W projekcie zaimplementowano mechanizmy ochrony przed atakami typu CSRF (Cross-Site Request Forgery), które polegają na wymuszeniu przez atakującego wykonania nieautoryzowanych żądań w imieniu zalogowanego użytkownika. Ataki tego typu mogą prowadzić do niepożądanych zmian w systemie, takich jak modyfikacja danych, wykonywanie transakcji lub uzyskanie dostępu do wrażliwych informacji.

Spring Security zapewnia wbudowane wsparcie dla ochrony przed atakami CSRF. W projekcie funkcja ta została aktywowana domyślnie, co powoduje generowanie unikalnych tokenów CSRF dla każdej sesji użytkownika. Tokeny te są dołączane do każdej chronionej akcji wykonywanej za pomocą metod POST, PUT, DELETE czy PATCH.

Działanie mechanizmu CSRF

- 1) **Generowanie tokenu CSRF** – Spring Security generuje unikalny token CSRF przy każdej sesji użytkownika.
- 2) **Przesyłanie tokenu** – Token CSRF jest automatycznie wstawiany do formularzy HTML przez mechanizm Thymeleaf lub inne szablony. Może być również przesyłany jako nagłówek w żądaniach AJAX.
- 3) **Weryfikacja tokenu** – Przy każdej akcji wymagającej modyfikacji danych serwer weryfikuje zgodność tokenu przesłanego w żądaniu z tym, który został wygenerowany dla bieżącej sesji użytkownika.

Mechanizm ochrony CSRF to kluczowy element bezpieczeństwa aplikacji internetowej. Dzięki automatycznemu generowaniu i weryfikacji tokenów przez Spring Security, projekt został zabezpieczony przed jednym z popularnych wektorów ataków, zwiększając tym samym bezpieczeństwo użytkowników i integralność danych.

7. PODSUMOWANIE

W ramach realizacji projektu prostego sklepu internetowego szczególną uwagę poświęcono bezpieczeństwu aplikacji, wdrażając szereg mechanizmów chroniących zarówno dane użytkowników, jak i integralność całego systemu. Wdrożono zaawansowane techniki, które zwiększają odporność aplikacji na ataki i błędy wynikające z niewłaściwego wprowadzania danych.

Hasła użytkowników są zabezpieczone przy użyciu algorytmu BCrypt, który zapewnia ich silne szyfrowanie i ochronę przed przechwyceniem lub odczytem w przypadku ewentualnego wycieku bazy danych. W procesie rejestracji zastosowano dodatkowe walidacje, takie jak weryfikacja dostępności loginu, konieczność dwukrotnego wpisania hasła, minimalna długość hasła oraz eliminacja niedozwolonych znaków, co minimalizuje ryzyko manipulacji danymi.

Przy logowaniu wdrożono mechanizmy zabezpieczające, takie jak Google reCAPTCHA v2, która aktywuje się po trzech nieudanych próbach logowania, skutecznie chroniąc aplikację przed atakami typu brute-force. Dodatkowo w formularzach, takich jak pole numeru telefonu, zadbano o odpowiednią walidację wprowadzanych danych, uniemożliwiając wpisywanie liter czy znaków specjalnych.

Mechanizm ochrony przed atakami typu CSRF (Cross-Site Request Forgery), wbudowany w Spring Security, chroni aplikację przed nieautoryzowanymi żądaniami, co dodatkowo wzmacnia jej odporność na popularne ataki sieciowe.

Podjęte kroki sprawiają, że aplikacja jest nie tylko funkcjonalna, ale przede wszystkim bezpieczna dla użytkowników. Zastosowane mechanizmy zabezpieczeń, walidacje danych oraz szyfrowanie haseł stanowią solidną podstawę do dalszego rozwoju projektu, jednocześnie spełniając współczesne standardy bezpieczeństwa w aplikacjach internetowych.