Laboratorium 10. *Spring*, *JWT* i *MySQL*

Cel zajęć

Realizacja zadań z laboratorium umożliwi studentom przygotowanie konfiguracji *Spring Security* z autentykacją za pomocą tokena *JWT*, podobnie jak w poprzednim laboratorium, ale z wykorzystaniem danych użytkowników przechowywanych w bazie danych *MySQL*.

Zakres tematyczny

- Utworzenie aplikacji z wykorzystaniem klas zdefiniowanych na laboratorium 9 tak, aby aplikacja korzystała z danych użytkowników przechowywanych w bazie *MySQL*.
- Konfiguracja *Spring Security*, definicja klas do pracy z danymi użytkownika z bazy danych, *i*mplementacja klasy kontrolera do obsługi autentykacji użytkowników tokenem *JWT*.
- Testowanie autentykacji w narzędziu *Postman*.

Proces konfiguracji *Spring Security*, generowania i walidacji tokena *JWT*, zrealizowano według przykładu prezentowanego na stronie [11]: https://www.techgeeknext.com/spring/spring-boot-security-token-authentication-jwt-mysql

Definicje klas prezentowanych w przykładach niniejszego laboratorium, niezbędne do prawidłowej konfiguracji *Spring Security* do pracy z tokenem *JWT* i bazą użytkowników w *MySQL* pochodzą w całości ze strony [11]. W celu lepszego zrozumienia metod prezentowanych klas, komentarze w kodach, zostały przetłumaczone na język polski.

Zadanie 10.1. *JWT* i *MySQL*

Wygeneruj nowy projekt o nazwie np. ***PAI_jwt*** za pomocą narzędzia *Spring Initializr* z dołączonymi zależnościami *Web, Security*, *JPA* i *MySQL* (Rys. 10.1).

Rys. 10.1. Tworzenie projektu w *Spring Initializr*

Do pliku ***pom.xml*** dodaj zależność dla *JWT*:

```
<dependency>
	<groupId>io.jsonwebtoken</groupId>
	<artifactId>jjwt</artifactId>
	<version>0.9.1</version>
</dependency>
```

Uruchom serwer *MySQL* i do pliku ***application.properties*** dodaj konfigurację, jak pokazuje Przykład 10.1.

Przykład 10.1. Plik application.properties

```
## secret key wykorzystywany przez algorytm haszujący,
# dodawany przez JWT do kombinacji z nagłówkiem (header)
# i ładunkiem (payload) z danymi
jwt.secret=Programowanie_ai2021
```

```
## Spring DATASOURCE (konfiguracja i właściwości źródła danych
spring.datasource.url =
jdbc:mysql://localhost:3306/notesdb?createDatabaseIfNotExist=true&allowP
ublicKeyRetrieval=true&useSSL=false
spring.datasource.username = root
spring.datasource.password =
spring.datasource.platform=mysql
spring.datasource.initialization-mode=always

## Właściwości dla Hibernate
# dialekt Hibernate
spring.jpa.properties.hibernate.dialect =
org.hibernate.dialect.MySQL8Dialect

# Ustawienia dla Hibernate dla operacji ddl
# (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = create-drop
```

Zbuduj projekt, a następnie do głównego pakietu dodaj podpakiety: ***config***, ***controller***, ***model***, ***repository*** i ***service***.

10.1.1. Klasy *UserDao* i *UserDto*

W pakiecie ***model*** utwórz klasę encji ***UserDao*** (Przykład 10.2) oraz klasę ***UserDto*** (Przykład 10.3). Klasa ***UserDto*** pobiera wartości użytkownika oraz przekazuje je do warstwy *DAO* w celu utrwalenia w bazie danych.

Przykład 10.2. Klasa UserDao

```
package bp.PAI_jwt.model;
import javax.persistence.*;
import net.minidev.json.annotate.JsonIgnore;

@Entity
@Table(name = "user")
public class UserDao {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column
    private String username;
    @Column
    @JsonIgnore
    private String password;

    public String getUsername() {
        return username;
    }
```

```
    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

Przykład 10.3. Klasa UserDto w pakiecie bp.PAI_jwt.model;

```
/* Klasa modelu UserDto odpowiada za pobranie wartości od użytkownika
   i przekazanie ich do warstwy DAO w celu wstawienia do bazy danych.
*/

public class UserDto {
    private String username;
    private String password;

    public String getUsername() {
        return username;
    }

    public void setUsername(String username) {
        this.username = username;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }
}
```

10.1.2. Repozytorium *JPA*

W pakiecie ***repository*** utwórz interfejs ***UserRepository*** rozszerzający repozytorium ***CrudRepository*** (Przykład 10.4). Repozytorium pozwala na dostęp do informacji o użytkownikach przechowywanych w bazie danych.

Przykład 10.4. Repozytorium UserRepository

```
package bp.PAI_jwt.repository;
import bp.PAI_jwt.model.UserDao;
import org.springframework.data.repository.CrudRepository;
```

```
public interface UserRepository extends CrudRepository<UserDao, Integer>
{
    UserDao findByUsername(String username);
}
```

10.1.3. Konfiguracja *Web Security*

Do pakietu ***config*** dodaj klasę ***WebSecurityConfig*** (Przykład 10.5), w której należy wskazać punkty końcowe ***/autheticate*** oraz ***/register***, niewymagające autentykacji.

W klasie zostały „wstrzyknięte" obiekty klas ***JwtAuthenticationEntryPoint***, ***JwtUserDetailsService*** i ***JwtRequestFilter***, które zostaną zdefiniowane w dalszych punktach.

Przykład 10.5. Klasa WebSecurityConfig [11]

```
package bp.PAI_jwt.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.config.annotation.authentication.builders.A
uthenticationManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.Enab
leGlobalMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity
;
import
org.springframework.security.config.annotation.web.configuration.EnableW
ebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecu
rityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenti
cationFilter;

@Configuration
@EnableWebSecurity
```

```
@EnableGlobalMethodSecurity(prePostEnabled = true)

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired
private JwtAuthenticationEntryPoint jwtAuthenticationEntryPoint;

@Autowired
private UserDetailsService jwtUserDetailsService;

@Autowired
private JwtRequestFilter jwtRequestFilter;

@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth)
          throws Exception {
  // Konfiguracja menadżera AuthenticationManager, aby wiedział skąd
  // pobrać dane użytkownika do sprawdzenia
  // Haszowania hasła BCryptPasswordEncoder
  auth.userDetailsService(jwtUserDetailsService).
  passwordEncoder(passwordEncoder());
}

@Bean
public PasswordEncoder passwordEncoder() {
       return new BCryptPasswordEncoder();
}

@Bean
@Override
public AuthenticationManager authenticationManagerBean()
         throws Exception {
       return super.authenticationManagerBean();
}

@Override
protected void configure(HttpSecurity httpSecurity) throws Exception {
  // W przykładzie nie ma potrzeby stosowania zabezpieczenia przed CSRF
  httpSecurity.csrf().disable()
   // poniższe żądanie nie wymaga uwierzytelniania
   .authorizeRequests().antMatchers("/authenticate", "/register")
   .permitAll()
   // wszystkie pozostałe żądania wymagają uwierzytelniania
   .anyRequest().authenticated().and()
   // sesja jest bezstanowa, nie przechowuje stanu użytkownika
   .exceptionHandling()
   .authenticationEntryPoint(jwtAuthenticationEntryPoint)
   .and().sessionManagement()
   .sessionCreationPolicy(SessionCreationPolicy.STATELESS);

  // Dodanie filtra w celu walidacji tokena dla każdego żądania
```

```
  httpSecurity.addFilterBefore(jwtRequestFilter,
               UsernamePasswordAuthenticationFilter.class);
 }
}
```

Do odpowiednich pakietów projektu dodaj klasy ***JwtRequestFilter, JwtTokenUtil, JwtAuthenticationEntryPoint*** (zdefiniowane w zadaniach z laboratorium 9) (Rys. 10.2).

10.1.4. Implementacja klasy *JwtAuthenticationController*

Do pakietu ***model*** dodaj definicje 2 klas zdefiniowanych w poprzednim laboratorium: ***JwtRequest*** i ***JwtResponse*** (Rys. 10.2). Utwórz klasę kontrolera ***JwtAuthenticationController*** z punktami końcowymi do autentykacji i rejestracji użytkownika (Przykład 10.6).

Przykład 10.6. Klasa JwtAuthenticationController [11]

```
package bp.PAI_jwt.config;

import bp.PAI_jwt.config.JwtTokenUtil;
import bp.PAI_jwt.model.JwtRequest;
import bp.PAI_jwt.model.JwtResponse;
import bp.PAI_jwt.model.UserDto;
import bp.PAI_jwt.service.JwtUserDetailsService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.DisabledException;
import
org.springframework.security.authentication.UsernamePasswordAuthenticati
onToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.*;

@RestController
@CrossOrigin
public class JwtAuthenticationController {
    @Autowired
    private AuthenticationManager authenticationManager;
    @Autowired
    private JwtTokenUtil jwtTokenUtil;
    @Autowired
    private JwtUserDetailsService userDetailsService;
```

```
    @RequestMapping(value = "/authenticate",
                    method = RequestMethod.POST)
    public ResponseEntity<?> createAuthenticationToken(@RequestBody
                    JwtRequest authenticationRequest) throws Exception {
        authenticate(authenticationRequest.getUsername(),
                     authenticationRequest.getPassword());
        final UserDetails userDetails =
           userDetailsService.loadUserByUsername(authenticationRequest
          .getUsername());
        final String token = jwtTokenUtil.generateToken(userDetails);
        return ResponseEntity.ok(new JwtResponse(token));
    }

    @RequestMapping(value = "/register", method = RequestMethod.POST)
    public ResponseEntity<?> saveUser(@RequestBody UserDto user)
           throws Exception {
        return ResponseEntity.ok(userDetailsService.save(user));
    }

    private void authenticate(String username, String password)
            throws Exception {
        try {
          authenticationManager.authenticate(new
             UsernamePasswordAuthenticationToken(username, password));
        } catch (DisabledException e) {
            throw new Exception("USER_DISABLED", e);
        } catch (BadCredentialsException e) {
            throw new Exception("INVALID_CREDENTIALS", e);
        }
    }
}
```

Zwróć uwagę na adnotację ***@CrossOrigin*** poprzedzającą klasę kontrolera.

CORS (ang. *Cross-Origin Resource Sharing*) to mechanizm bezpieczeństwa, który wykorzystuje dodatkowe nagłówki *HTTP*, aby poinformować przeglądarkę, czy ma udostępnić dane zwrócone klientowi. Serwer decyduje, czy klient jest klientem zaufanym i na tej podstawie ustawia odpowiednie nagłówki, dzięki którym przeglądarka wie, czy ma udostępnić dane klientowi. W ten sposób przeglądarka zabezpiecza użytkownika przed m.in. atakami typu *CrossSite Request Forgery* (atak *CSRF* – polega na wysyłaniu w imieniu klienta żądań *HTTP* do złośliwych serwisów, wykorzystując dane użytkownika np. sesje, ciasteczka, stan zalogowania itp.).

Obsługa *CORS* w *Spring* jest możliwa za pomocą adnotacji ***@CrossOrigin*** poprzedzającej metodę lub klasę kontrolera. Ta adnotacja zezwala przeglądarce udostępniać dane pochodzące ze wszystkich źródeł (ang. *origin*). Czas życia

odpowiedzi (ang. *response*) jest utrzymywany w cache przeglądarki przez 30 minut (domyślna wartość parametru *maxAge*).

W celu przetestowania aplikacji, do pakietu ***controller*** dodaj klasę ***TestController*** jak w poprzednim laboratorium (Rys. 10.2). Klasę ***TestController*** także poprzedź adnotacją ***@CrossOrigin***.

Struktura plików gotowego projektu przedstawiona jest na Rys. 10.2.

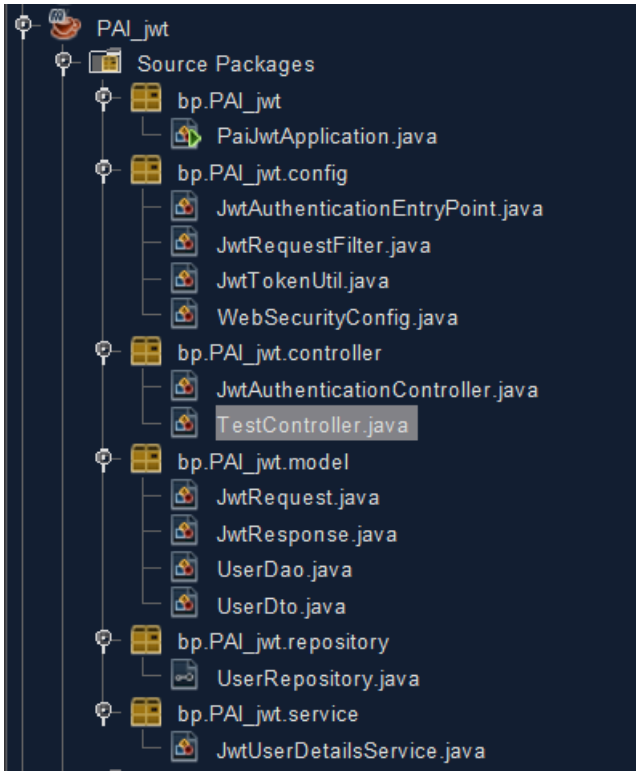
UWAGA! Jeśli korzystasz z wersji *Java* większej niż 8, budowa projektu zakończy się niepowodzeniem. W tym wypadku do pliku ***pom.xml*** dodaj zależność:

```
<dependency>
     <groupId>javax.xml.bind</groupId>
     <artifactId>jaxb-api</artifactId>
</dependency>
```

Jeśli nadal jest problem, do pliku ***application.properties*** dodaj wiersz:

```
spring.main.allow-circular-references=true
```

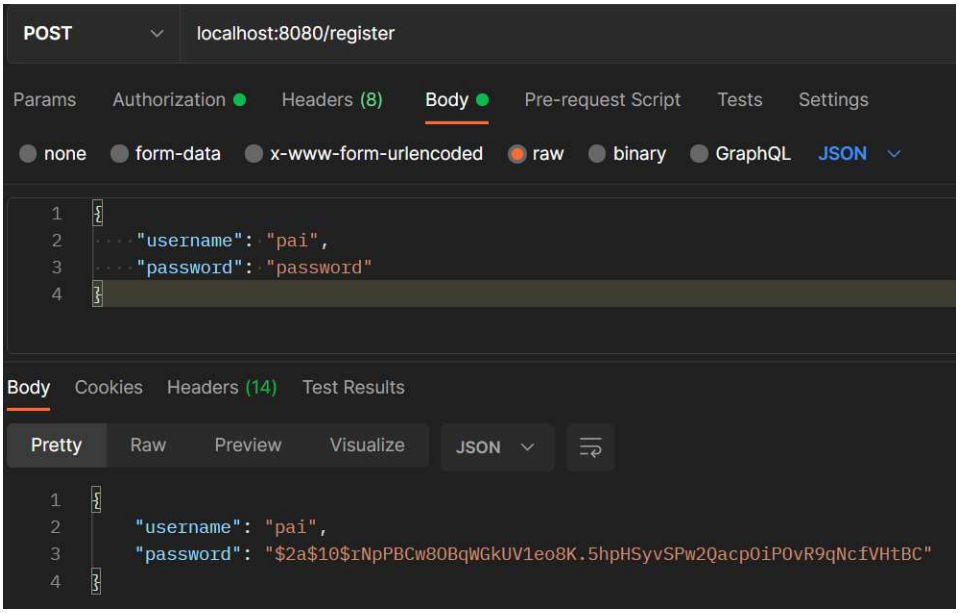
Problem może wynikać z faktu, że od wersji 2.6 w *Spring Boot* domyślnie zabronione są referencje cykliczne (ang. *Circular References Prohibited by Default in Spring Boot version 2.6*).



Rys. 10.2. Struktura projektu z *JWT* i *MySQL*

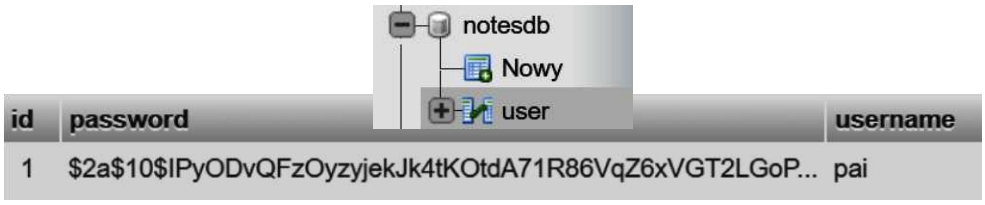
Zadanie 10.2. Test działania w narzędziu *Postman*

Przetestuj działanie aplikacji, wysyłając na początek żądanie metodą *POST* z danymi użytkownika pod adres ***/register*** (Rys. 10.3). W odpowiedzi zostanie zwrócony obiekt ***user*** w formacie *JSON* z zahaszowanym hasłem (Rys. 10.3).



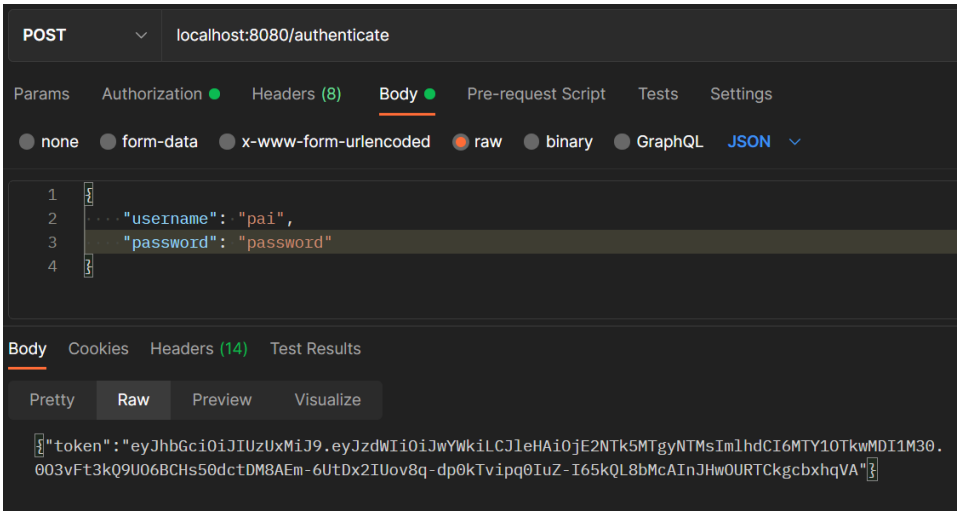
Rys. 10.3. Rejestracja danych użytkownika w bazie *MySQL*

Po prawidłowym wykonaniu akcji, powinna się utworzyć baza danych ***notesdb*** z tabelą ***user***, a w niej pierwszy rekord z danymi użytkownika (Rys. 10.4).



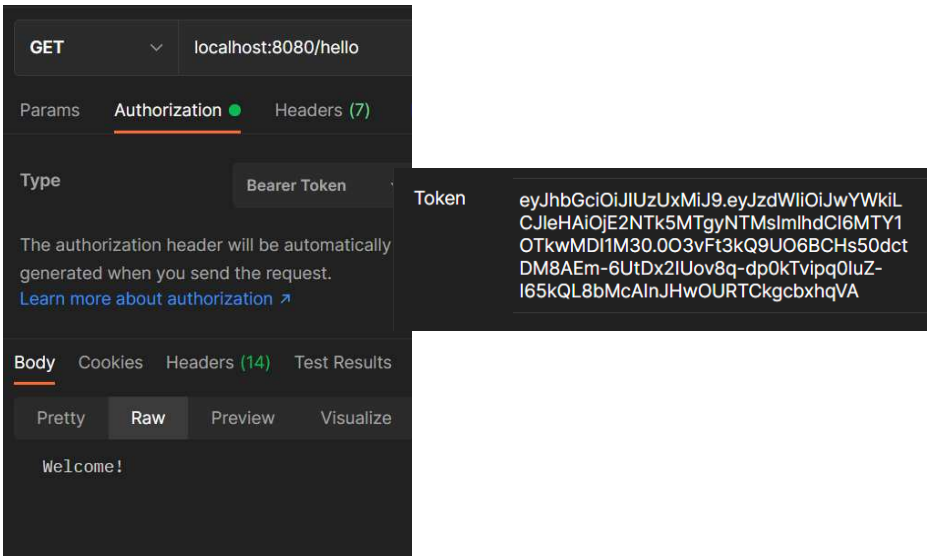
Rys. 10.4. Rekord w tabeli *user* w bazie *notesdb*

Metodą *POST* wyślij żądanie uwierzytelniające użytkownika (na podstawie danych zapisanych w bazie) pod adres ***/authenticate*** (Rys. 10.5). W odpowiedzi, po prawidłowym uwierzytelnieniu użytkownika, serwer powinien zwrócić token *JWT* (Rys. 10.5), analogicznie jak w laboratorium 9.



Rys. 10.5. Żądanie z przekazaniem danych użytkownika i odpowiedź z tokenem *JWT*

Po uzyskaniu tokenu, należy go dodać do nagłówka ***Authorization*** żądania, aby uzyskać dostęp do zawartości strony ***/hello*** (Rys. 10.6).



Rys. 10.6. Żądanie z przekazaniem tokena w nagłówku *Authorization*