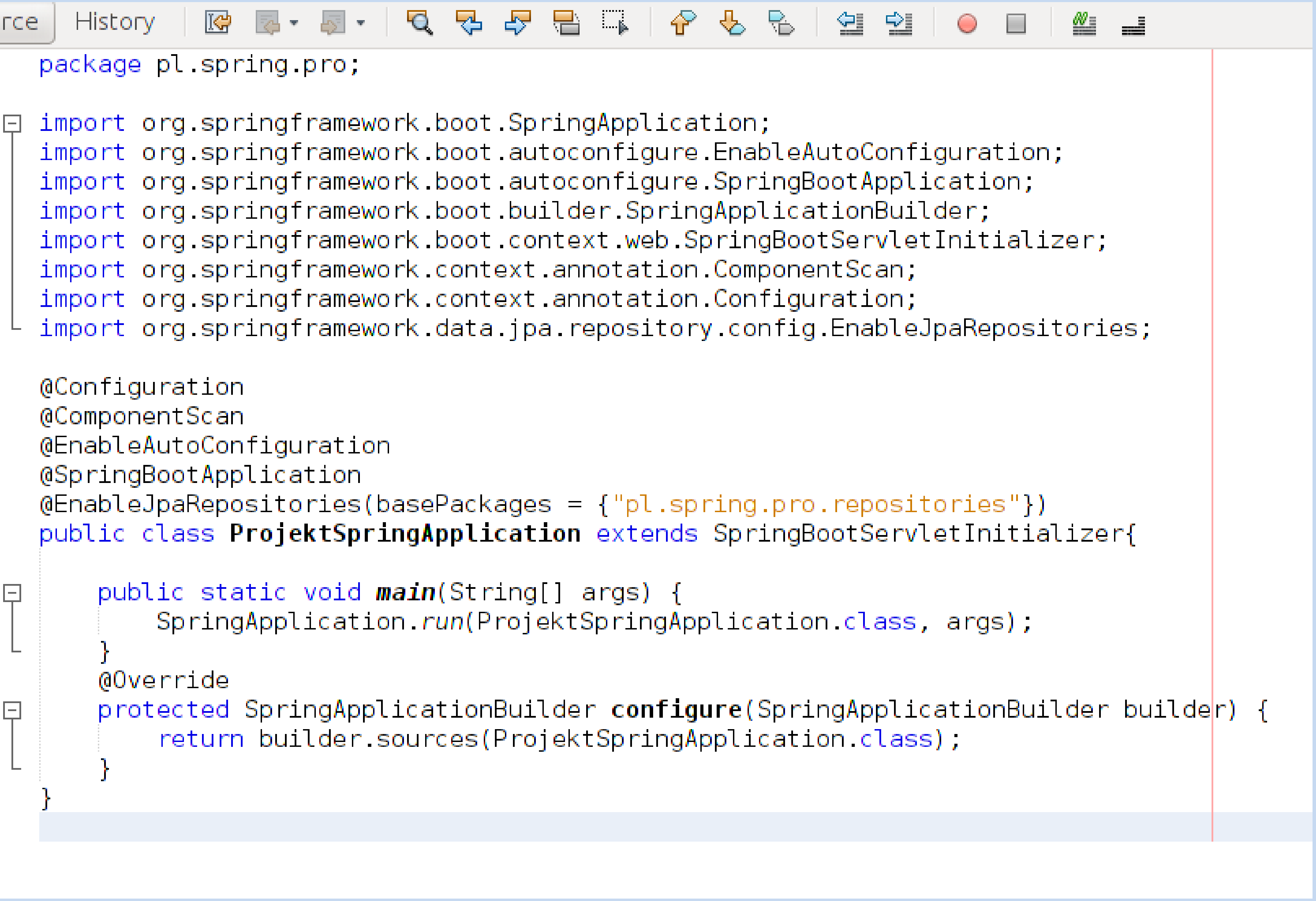
Spring Security



```
package pl.spring.pro;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.context.web.SpringBootServletInitializer;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@Configuration
@ComponentScan
@EnableAutoConfiguration
@SpringBootApplication
@EnableJpaRepositories(basePackages = {"pl.spring.pro.repositories"})
public class ProjektSpringApplication extends SpringBootServletInitializer{

    public static void main(String[] args) {
        SpringApplication.run(ProjektSpringApplication.class, args);
    }
    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
        return builder.sources(ProjektSpringApplication.class);
    }
}
```

Co to jest?/Do czego służy?

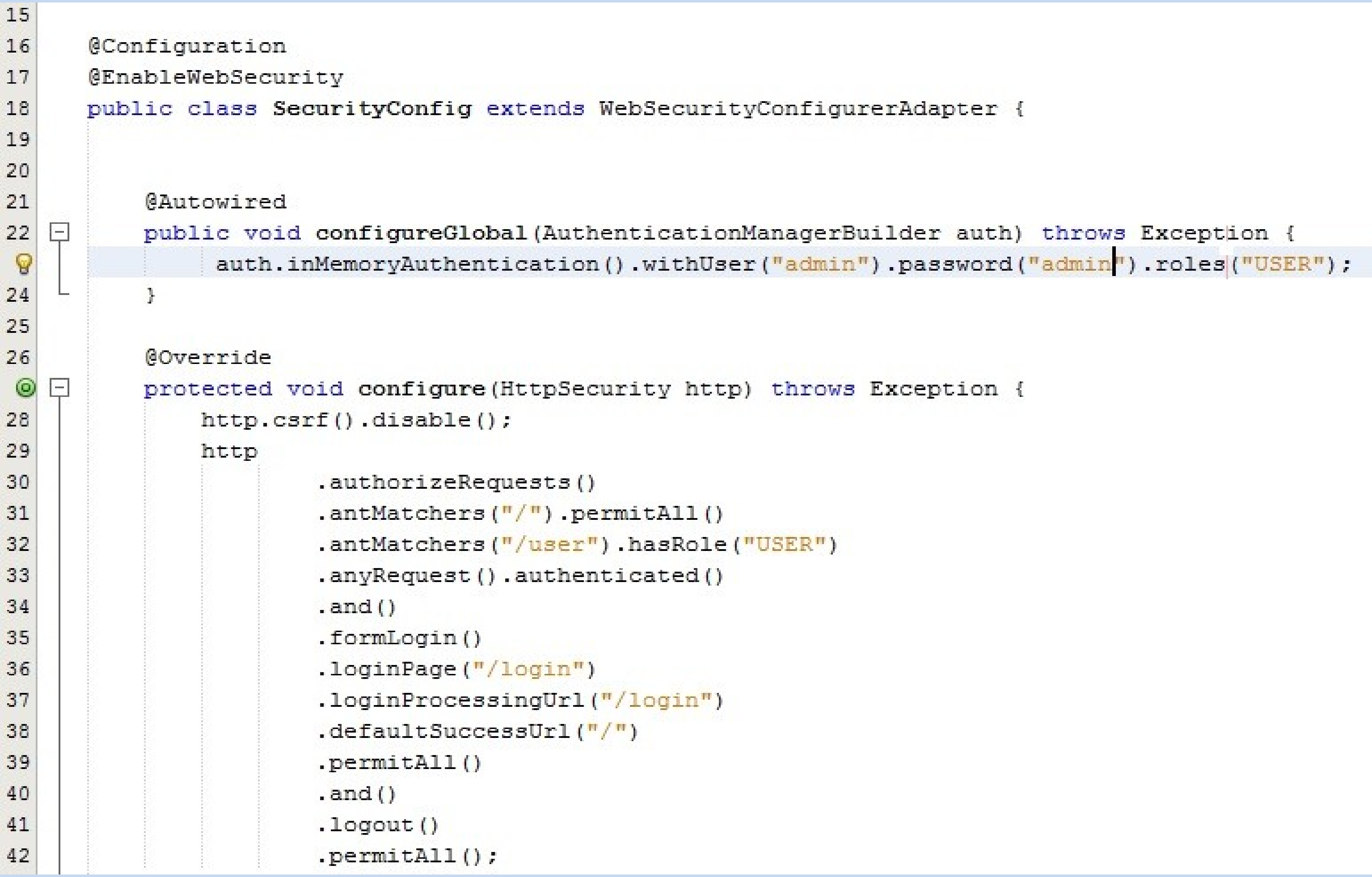
- Jest to część biblioteki Spring
- Zapewnia kompleksowe usługi bezpieczeństwa dla aplikacji Java Enterprises

https://docs.spring.io/spring-security/site/docs/3.0.x/reference/introduction.html#what-is-acegi-security

Konfiguracja

Maven-zależności

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
<dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <version>9.4-1201-jdbc41</version>
            <scope>runtime</scope>
        </dependency>
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
```



```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {


    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.inMemoryAuthentication().withUser("admin").password("admin").roles("USER");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
        http
                .authorizeRequests()
                .antMatchers("/").permitAll()
                .antMatchers("/user").hasRole("USER")
                .anyRequest().authenticated()
                .and()
                .formLogin()
                .loginPage("/login")
                .loginProcessingUrl("/login")
                .defaultSuccessUrl("/")
                .permitAll()
                .and()
                .logout()
                .permitAll();
```

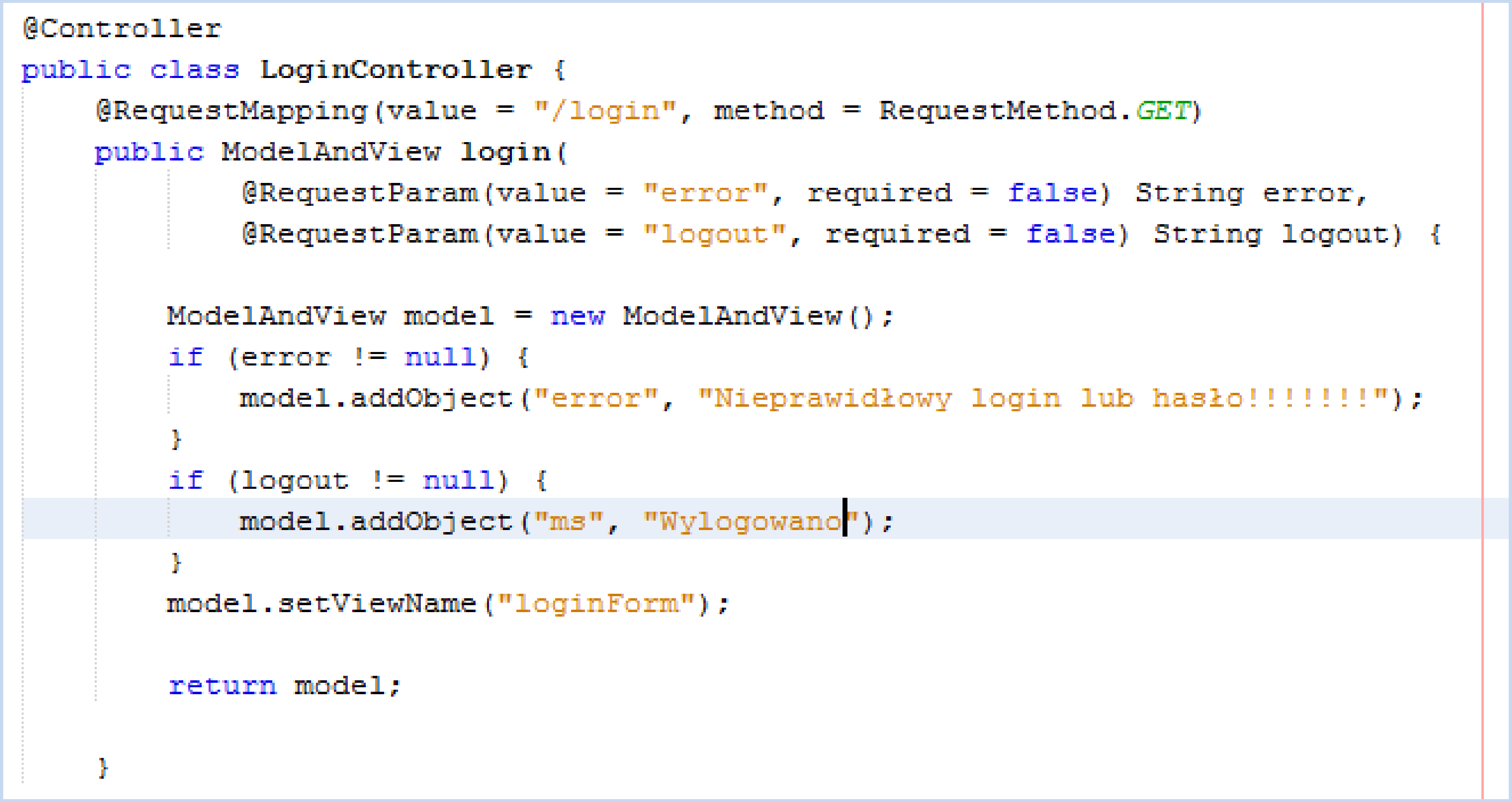
Parametry http w metodzie configure

```
.authorizeRequests()                                   \\ specjalnie wnioski
            .antMatchers("/").permitAll()              \\ zezwól wszystkim
            .antMatchers("/user").hasRole("USER")  \\ zezwól tylko z rolą USER
.anyRequest().authenticated()                          \\ każdy uwierzytelniony
```

```
 .formLogin()
            .loginPage("/login")
            .loginProcessingUrl("/login")
            .defaultSuccessUrl("/")
            .permitAll()
            .and()
            .logout()
            .permitAll();
```

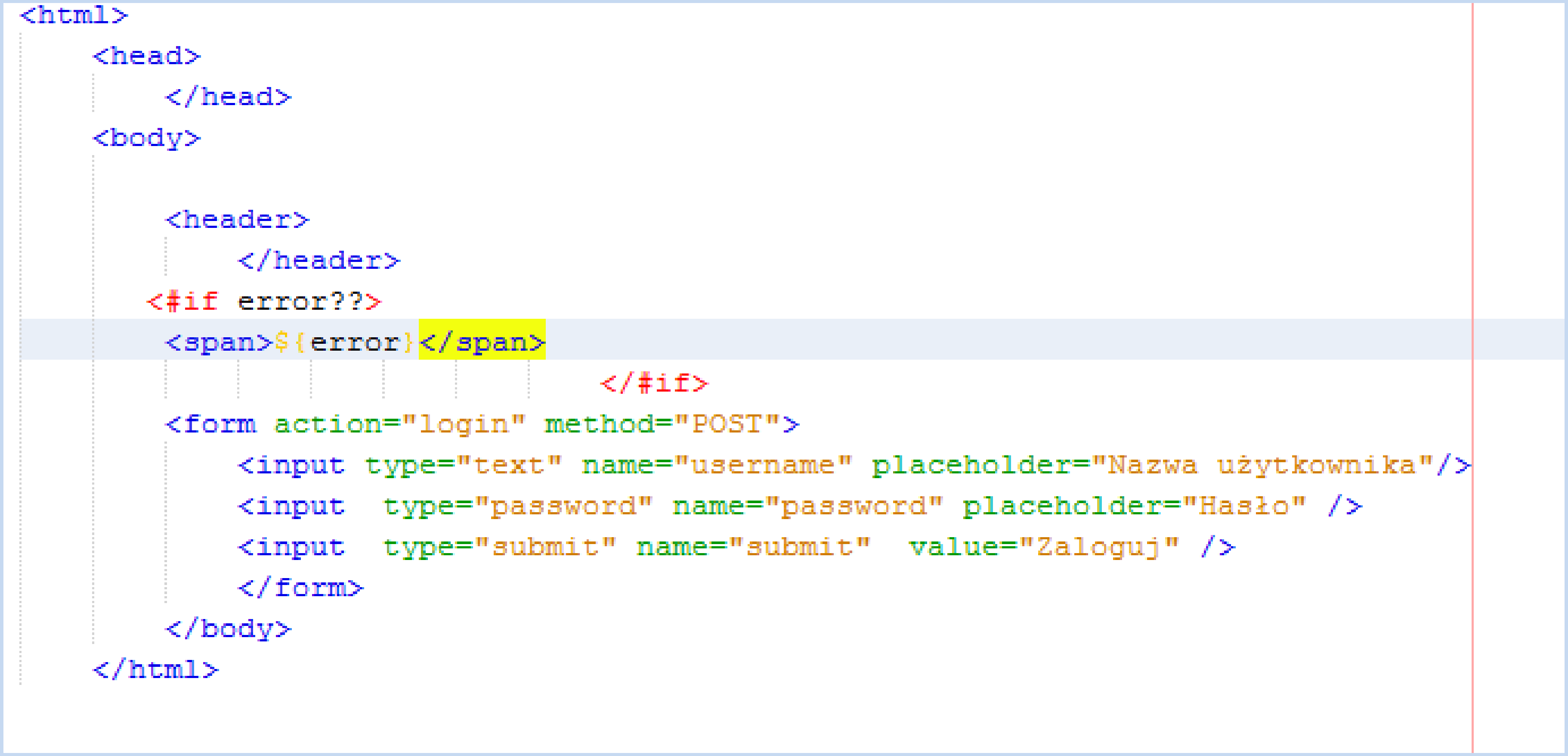
Konfiguracja logowania i wylogowywania się

Kontroler logowania



```
@Controller
public class LoginController {
    @RequestMapping(value = "/login", method = RequestMethod.GET)
    public ModelAndView login(
            @RequestParam(value = "error", required = false) String error,
            @RequestParam(value = "logout", required = false) String logout) {

        ModelAndView model = new ModelAndView();
        if (error != null) {
            model.addObject("error", "Nieprawidłowy login lub hasło!!!!!!!");
        }
        if (logout != null) {
            model.addObject("ms", "Wylogowano");
        }
        model.setViewName("loginForm");

        return model;

    }
```

Szablon logowania

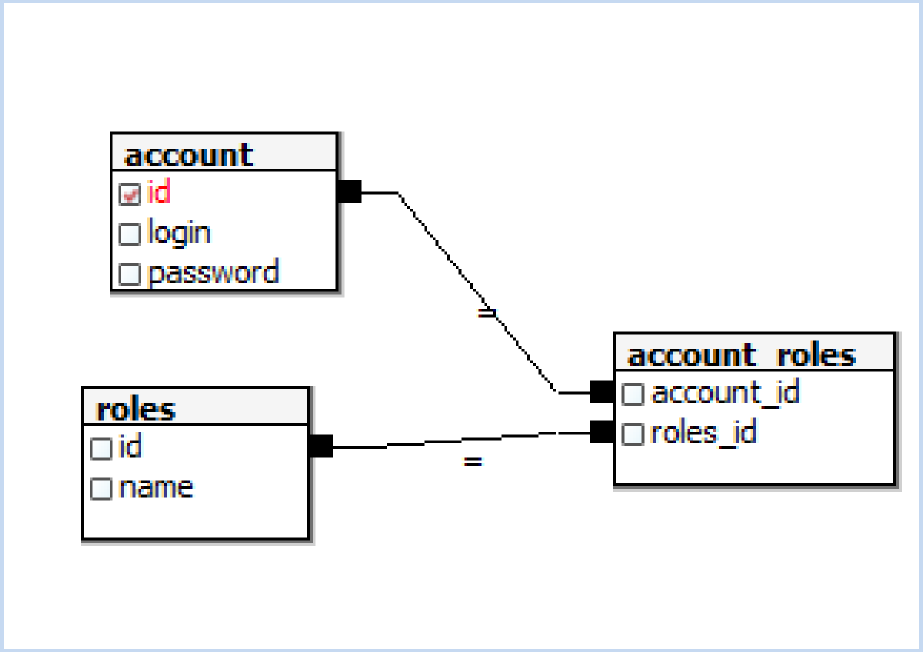


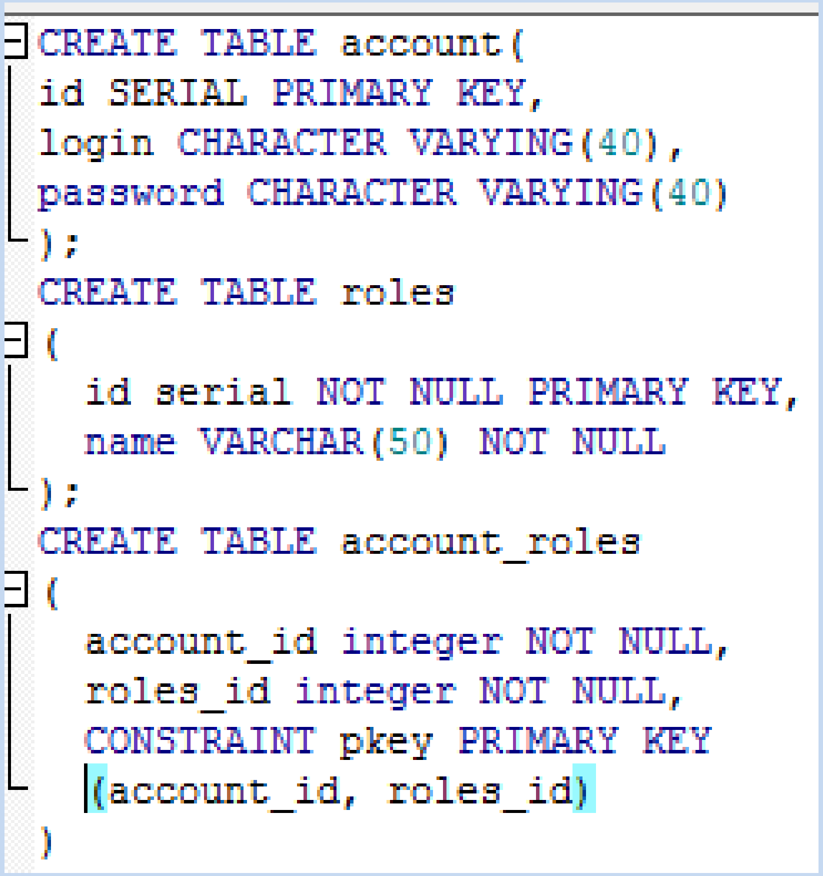
```
<html>
    <head>
        </head>
    <body>

        <header>
            </header>
      <#if error??>
        <span>${error}</span>
                                <#/if>
        <form action="login" method="POST">
            <input type="text" name="username" placeholder="Nazwa użytkownika"/>
            <input  type="password" name="password" placeholder="Hasło" />
            <input  type="submit" name="submit"  value="Zaloguj" />
            </form>
        </body>
    </html>
```

Konfiguracja z użyciem Bazy danych

Na przykładzie Postgres-a

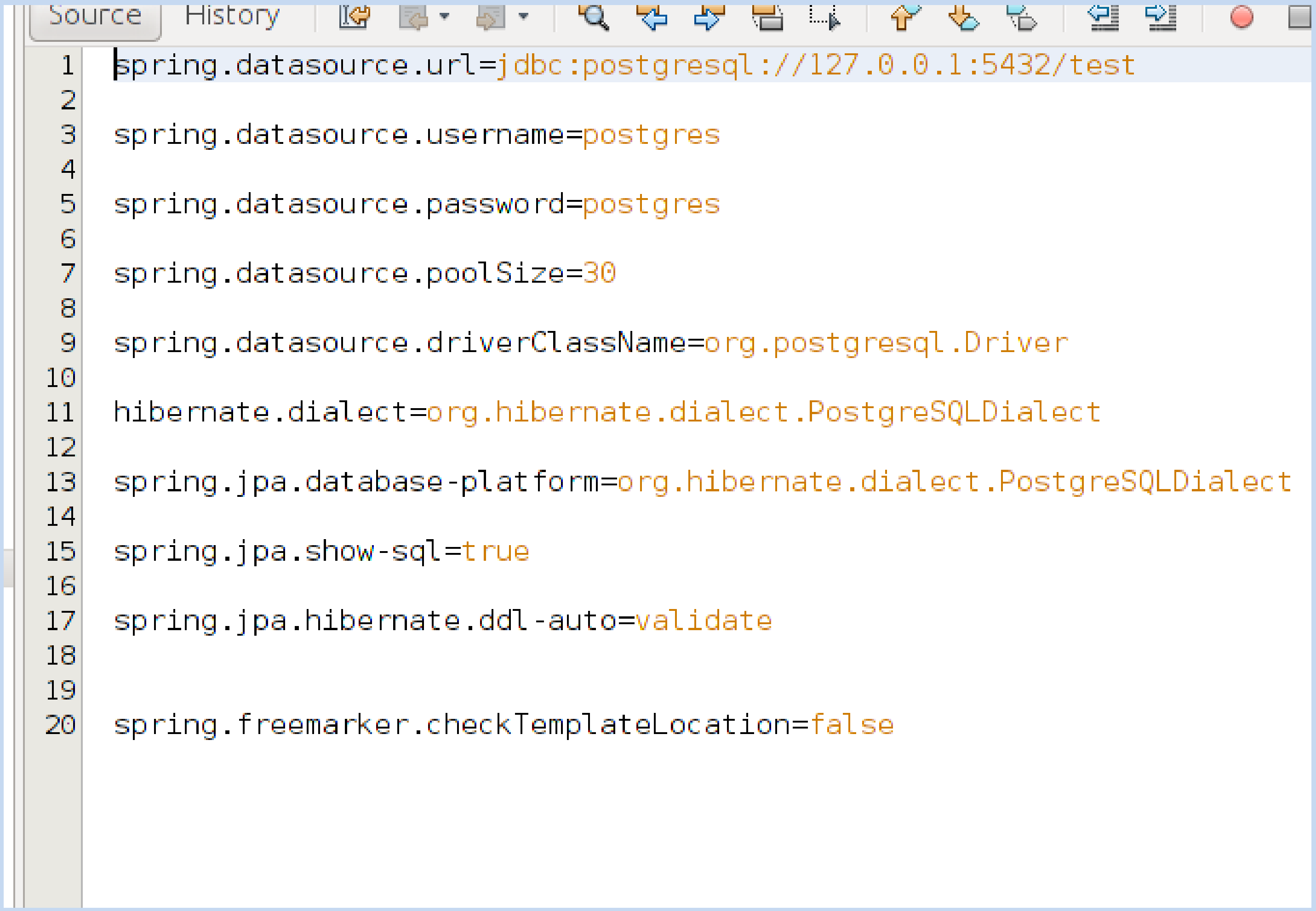
Baza danych





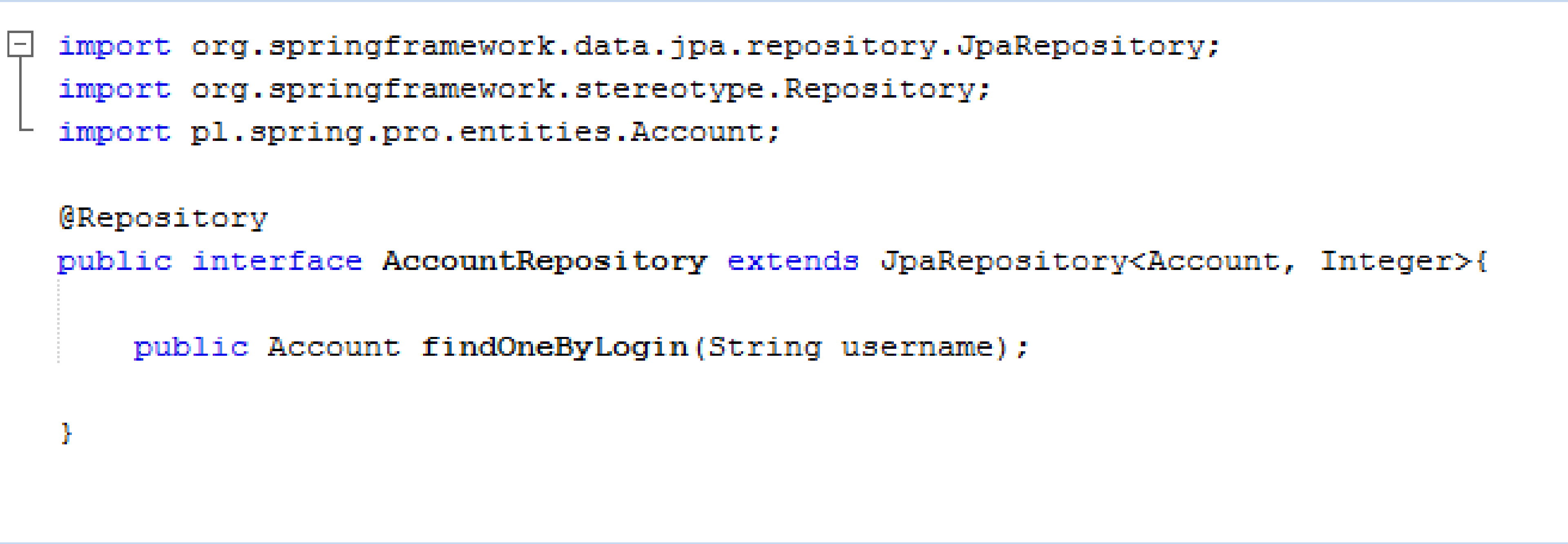
```
CREATE TABLE account(
id SERIAL PRIMARY KEY,
login CHARACTER VARYING(40),
password CHARACTER VARYING(40)
);
CREATE TABLE roles
(
  id serial NOT NULL PRIMARY KEY,
  name VARCHAR(50) NOT NULL
);
CREATE TABLE account_roles
(
  account_id integer NOT NULL,
  roles_id integer NOT NULL,
  CONSTRAINT pkey PRIMARY KEY
  (account_id, roles_id)
)
```

Konfiguracja Postgresa w JPA



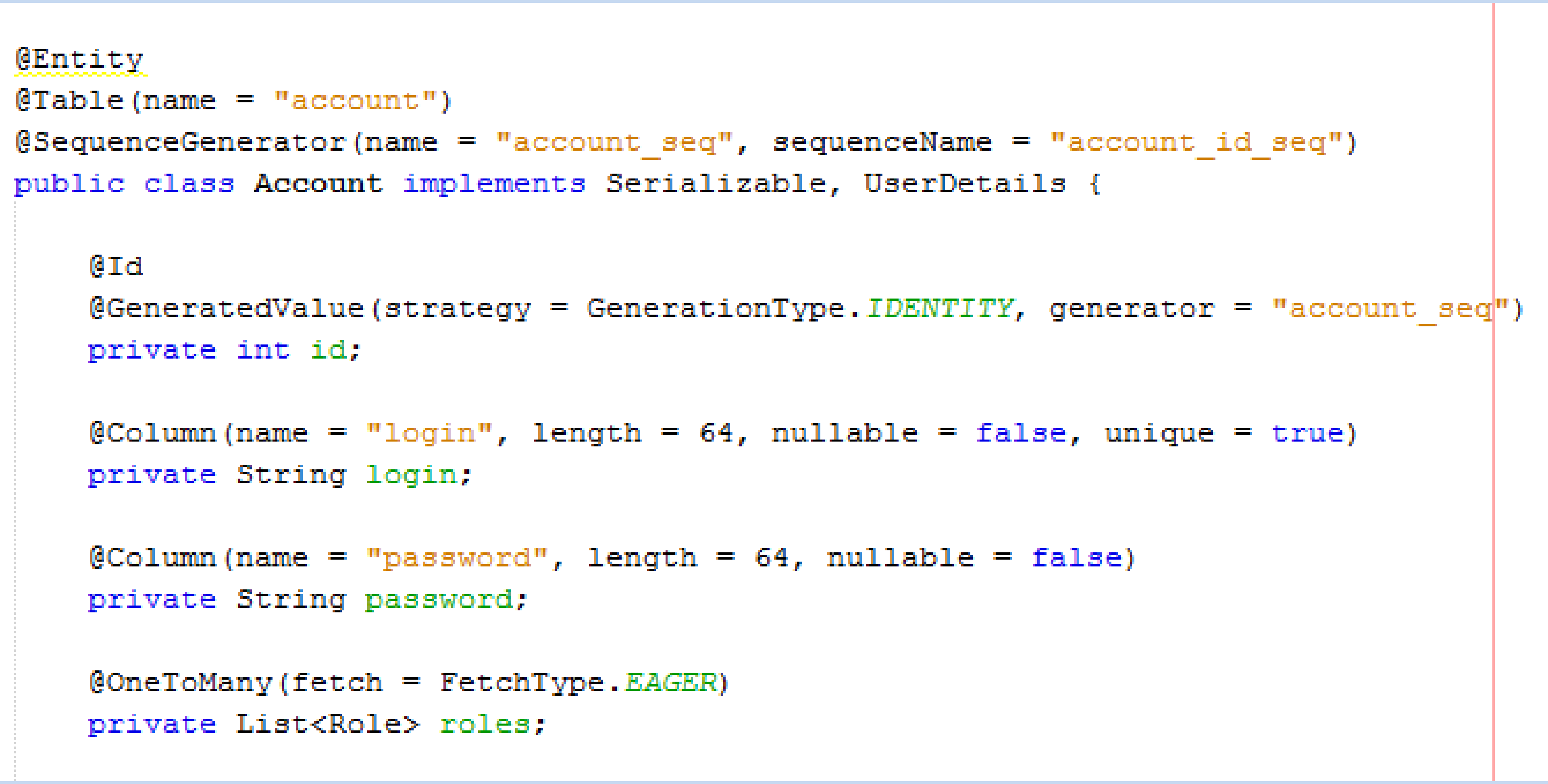
```
spring.datasource.url=jdbc:postgresql://127.0.0.1:5432/test

spring.datasource.username=postgres

spring.datasource.password=postgres

spring.datasource.poolSize=30

spring.datasource.driverClassName=org.postgresql.Driver

hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=validate


spring.freemarker.checkTemplateLocation=false
```

Połączenie z JPA



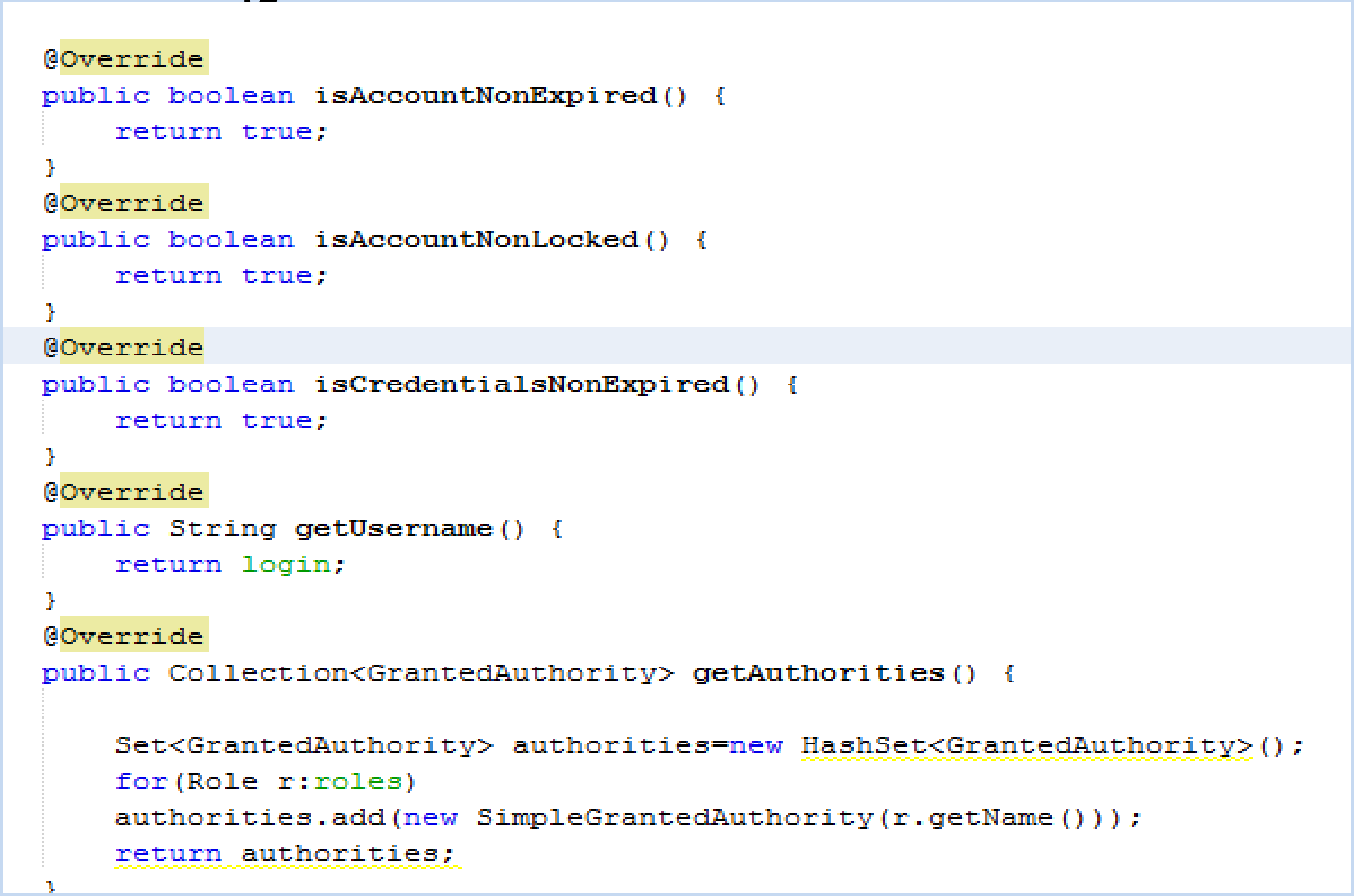
```java
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import pl.spring.pro.entities.Account;

@Repository
public interface AccountRepository extends JpaRepository<Account, Integer>{

    public Account findOneByLogin(String username);

}
```

Encja tabeli account



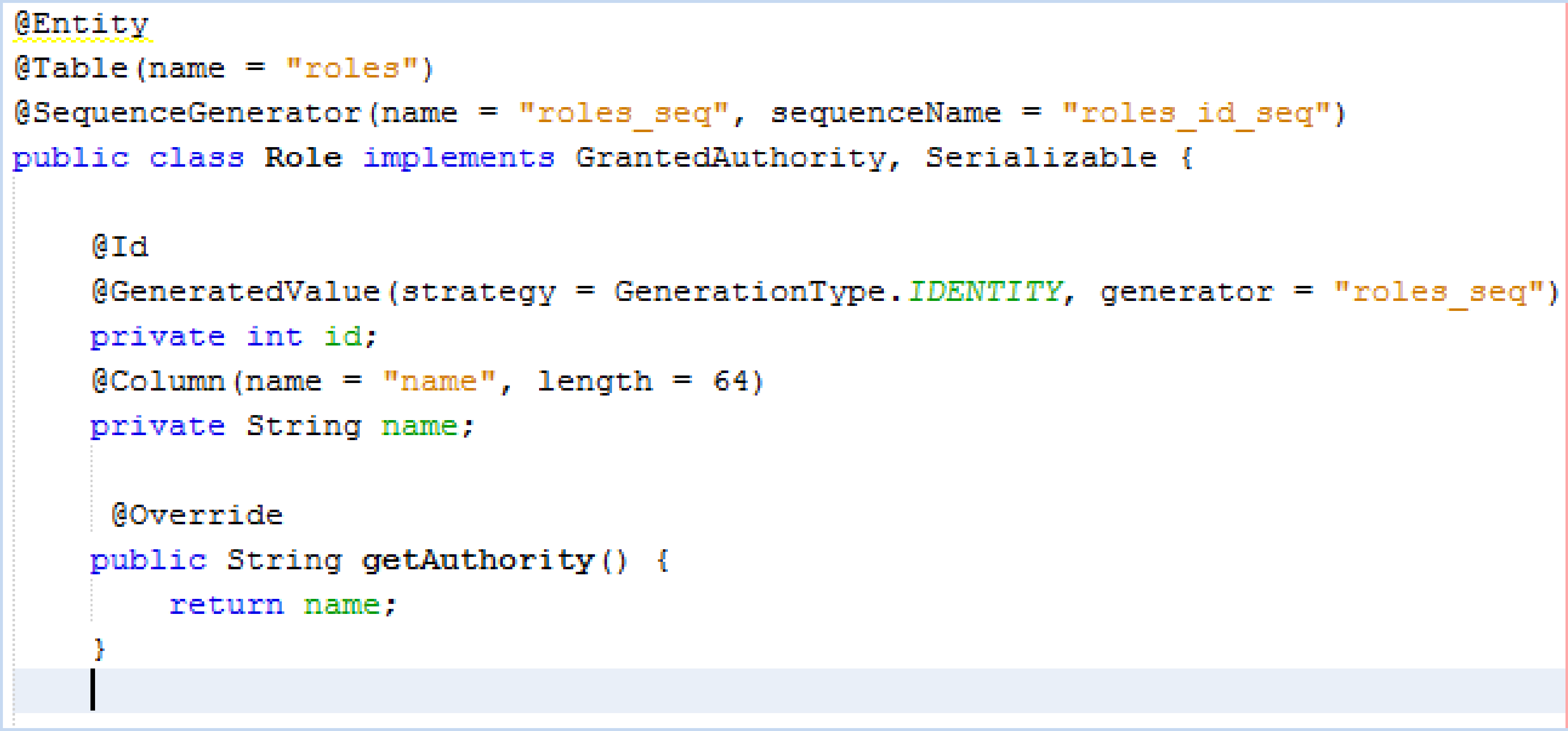
```
@Entity
@Table(name = "account")
@SequenceGenerator(name = "account_seq", sequenceName = "account_id_seq")
public class Account implements Serializable, UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "account_seq")
    private int id;

    @Column(name = "login", length = 64, nullable = false, unique = true)
    private String login;

    @Column(name = "password", length = 64, nullable = false)
    private String password;

    @OneToMany(fetch = FetchType.EAGER)
    private List<Role> roles;
```

Encja tabeli account cd



```
@Override
public boolean isAccountNonExpired() {
    return true;
}
@Override
public boolean isAccountNonLocked() {
    return true;
}
@Override
public boolean isCredentialsNonExpired() {
    return true;
}
@Override
public String getUsername() {
    return login;
}
@Override
public Collection<GrantedAuthority> getAuthorities() {

    Set<GrantedAuthority> authorities=new HashSet<GrantedAuthority>();
    for(Role r:roles)
    authorities.add(new SimpleGrantedAuthority(r.getName()));
    return authorities;
}
```

Encja tabeli roles



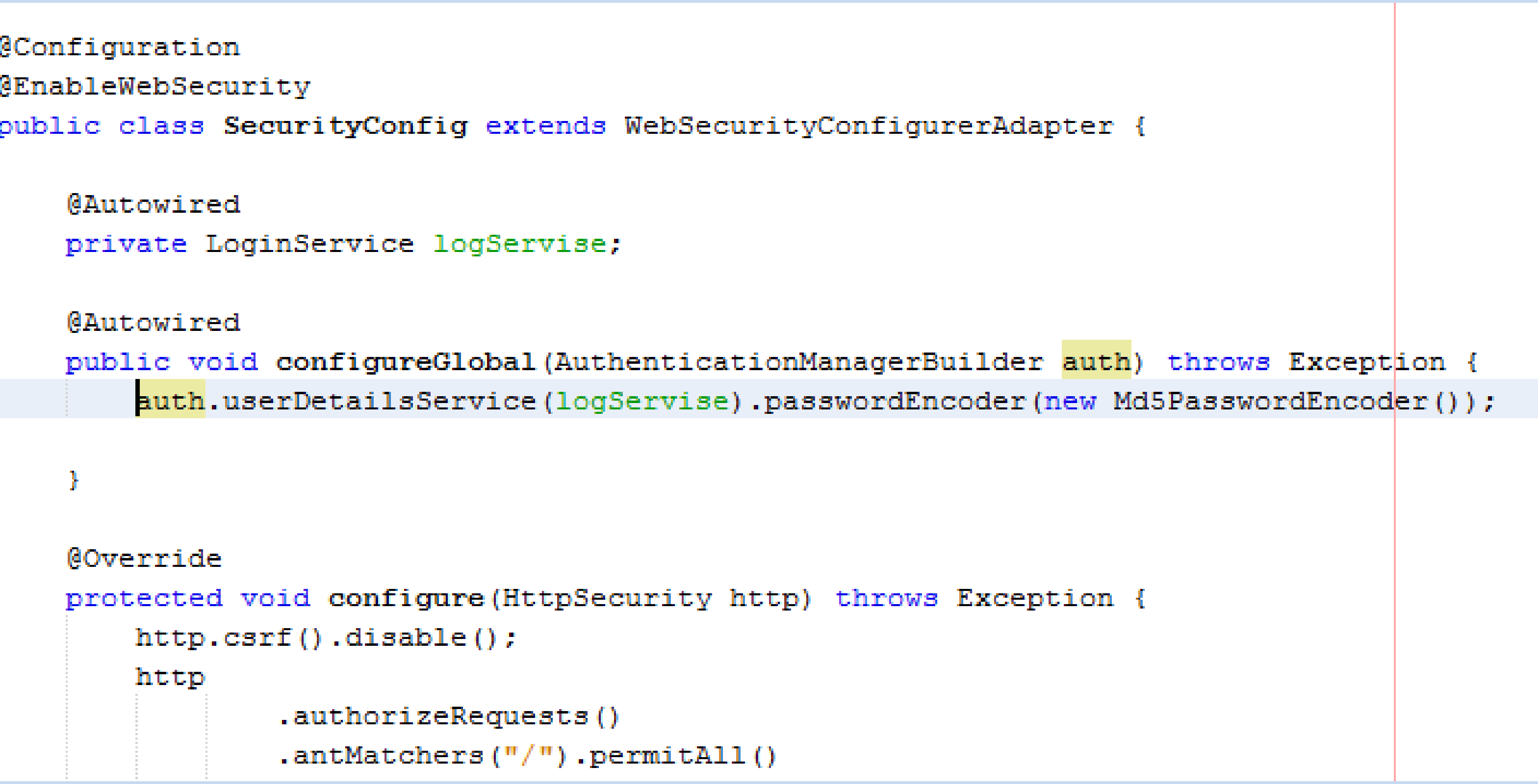
```
@Entity
@Table(name = "roles")
@SequenceGenerator(name = "roles_seq", sequenceName = "roles_id_seq")
public class Role implements GrantedAuthority, Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY, generator = "roles_seq")
    private int id;
    @Column(name = "name", length = 64)
    private String name;

     @Override
    public String getAuthority() {
        return name;
    }

```

Poza tym co widać w obu klasach encji należy wygenerować metody:

- equals
- hashCode

W Nedbeans: ALT+INSERT   equals and hashCode

Konfiguracja Zaawansowana

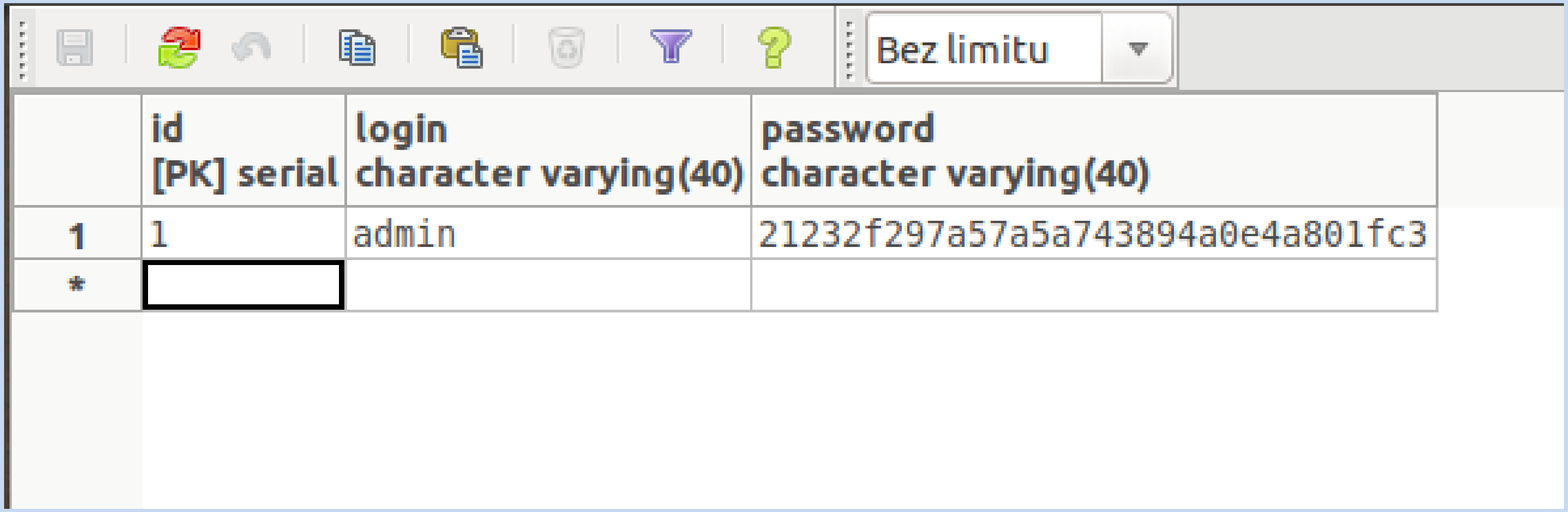


```
@Configuration
@EnableWebSecurity
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private LoginService logServise;

    @Autowired
    public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(logServise).passwordEncoder(new Md5PasswordEncoder());

    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
        http
                .authorizeRequests()
                .antMatchers("/").permitAll()
```

Parametry AuthenticationManagerBuilder

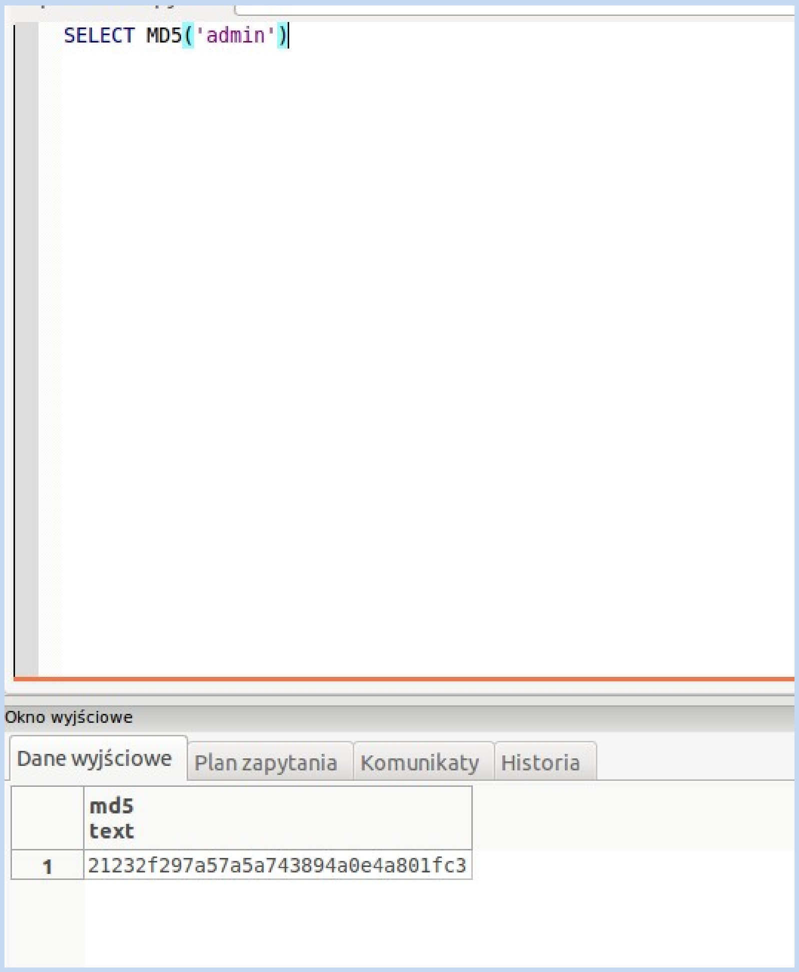
- UserDetailsService-określenie serwisu odpowiadającego za logowanie
- PasswordEncoder-określenie klasy szyfrowania hasła

Widok danych w tabeli account



	id [PK] serial	login character varying(40)	password character varying(40)
1	1	admin	21232f297a57a5a743894a0e4a801fc3
*			

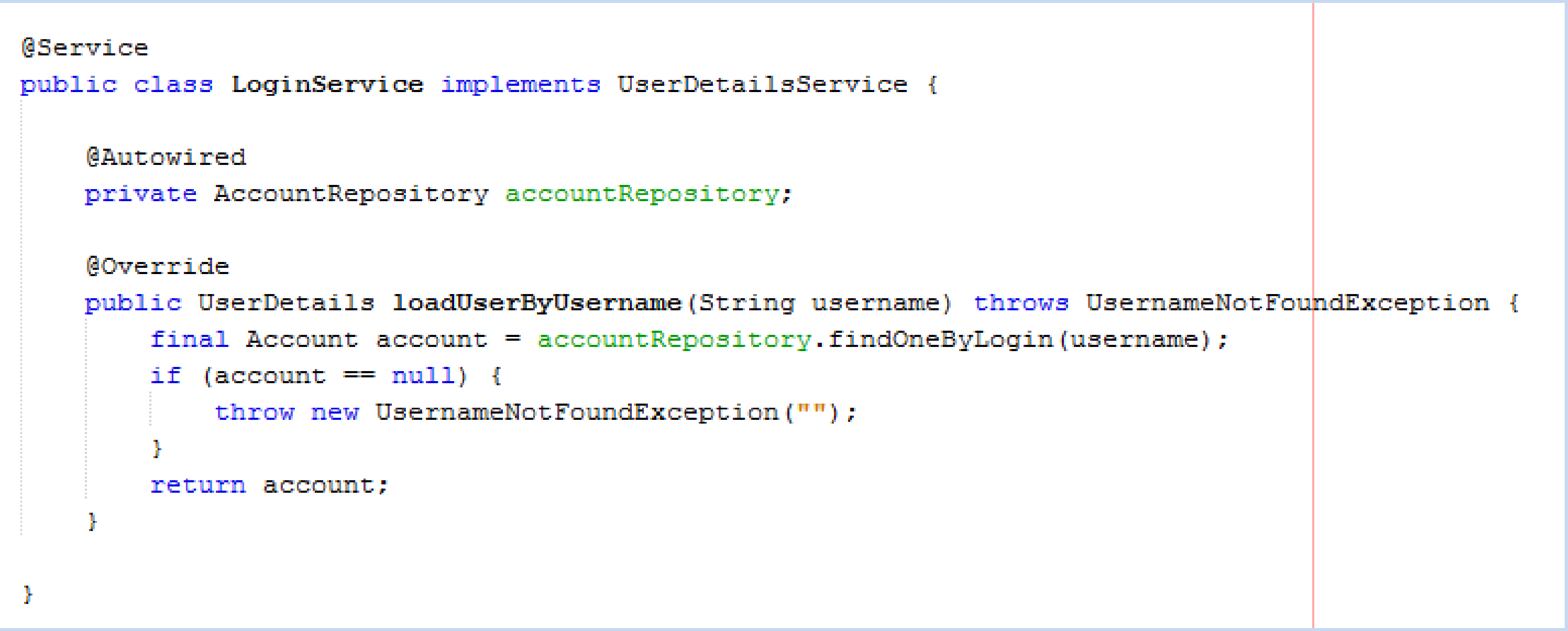
Uzyskiwanie zaszyfrowanego hasła



```
SELECT MD5('admin')
```

	md5 text
1	21232f297a57a5a743894a0e4a801fc3

Serwis pobierający dane



```java
@Service
public class LoginService implements UserDetailsService {

    @Autowired
    private AccountRepository accountRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        final Account account = accountRepository.findOneByLogin(username);
        if (account == null) {
            throw new UsernameNotFoundException("");
        }
        return account;
    }

}
```

Literatura

- Craig Walls „Spring w Akcji” Rozdział 9
- Willie Wheeler, Joshua White „Spring w praktyce” Rozdział 6 i 7
- http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/

KONIEC DZIĘKUJE ZA UWAGĘ