Tytuł projektu:

Portal ogłoszeń

„ZnajdzKlub”

Przedmiot: Aplikacje internetowe

Realizacja: Piotr Sikora

F1A-DU-L3

1. Założenia funkcjonalne aplikacji webowej

Aplikacji ma służyć do gromadzenia i publikacji ogłoszeń dotyczących różnego rodzaju klubów (np. sportowych, tanecznych).

Dodawanie klubów:

Chcąc dodać klub mamy do dyspozycji pola, w którym wybieramy rodzaj kategorii, wypełniamy tytuł(nazwę) klubu, dane adresowe oraz inne dodatkowe informacje.

Po dodaniu klubu musi być on zaakceptowany przez administratora w celu publikacji na portalu.

Filtrowanie klubów

Pomocną funkcją będzie filtrowanie klubów przy pomocy wpisania kodu pocztowego miasta oraz określenie obszaru, w którym będziemy szukać klubu.

Rejestracja użytkownika:

W celu dodania własnego klubu do list klubów na portalu należy się najpierw zarejestrować przechodząc na stronę z formularzem rejestracyjnym i poprawnie wypełniając dane.

Uprawnienia użytkowników:

Gość:

Przeglądanie ogłoszeń	Użytkownik może przeglądać wszystkie ogłoszenia zatwierdzone przez administratora

Standardowy użytkownik:

Edycja profilu	Użytkownik w każdej chwili może zmienić swoje dane w profilu
Usunięcie profilu	Użytkownik ma możliwość usunięcia swojego konta
Dodanie ogłoszenia	Zarejestrowany użytkownik może dodać ogłoszenie do wybranej kategorii
Edycja ogłoszenia	Użytkownik może edytować dodane przez siebie ogłoszenie
Usunięcie ogłoszenia	Użytkownik może usunąć dodane przez siebie ogłoszenie

Administrator:

Moderacja użytkowników	Możliwość edycji/usunięcia użytkownika
Zarządzanie kategoriami	Możliwość dodania/edycji/usunięcia kategorii ogłoszeń
Zarządzanie ogłoszeniami	Możliwość edycji/usunięcia wszystkich ogłoszeń

2. Założenia niefunkcjonalne aplikacji webowej

Podczas realizacji projektu wykorzystywana będzie technologia Java oparta na **wzorcu Spring MVC**, który jest szkieletem tworzenia aplikacji opartych na platformie Java EE. Spring jak sama nazwa wskazuje jest trójpowłokowy (Model, View, Controller), który pozwala na wysoki stopień kontroli nad szablonem poprzesz interfejsy np. JSP, lub FreeMarker.

W aplikacji będzie wykorzystywana baza danych **PostgreSQL**. Jest to jeden z najpopularniejszych darmowych systemów zarządzania relacyjnymi bazami danych.
Aby nawiązać połączenie bazy danych z aplikacją użyty zostanie **Hibernate**. Jest to framework do realizacji warstwy dostępu do danych i zapewnia on połączenie między relacyjna bazą danych a światem obiektowym.

Następnym narzędziem, który zostanie wykorzystany jest **Spring Security**. Jest to narzędzie niezbędne podczas uwierzytelniania i autoryzacji użytkownika pozwalające na precyzyjne i wygodne określenie reguł dla całych wzorców URL. Pozwala ograniczyć wyświetlanie np. wszystkich stron poza strona logowania.

Podczas budowy frontendu zastosowany będzie framework **AngularJS**. Jest to framework JavaScript stworzony przez Google służący do szybkiego i łatwego budowania aplikacji internetowych. Na dzień dzisiejszy pozwala on nawet na całkowite zbudowanie aplikacji bez korzystania z innych frameworków.

Aby nadać wygląd naszej aplikacji użyjemy biblioteki **Bootstrap**. Jest to framework CSS, który zawiera zestaw narzędzi, które ułatwiają tworzenie interfejsu graficznego naszej aplikacji.

Do prawidłowego działania portalu niezbędny jest serwer oraz baza danych. Serwer musi mieć możliwość hostingu aplikacji utworzonej w technologii Java oraz obsługiwać bazę danych PostgreSQL.

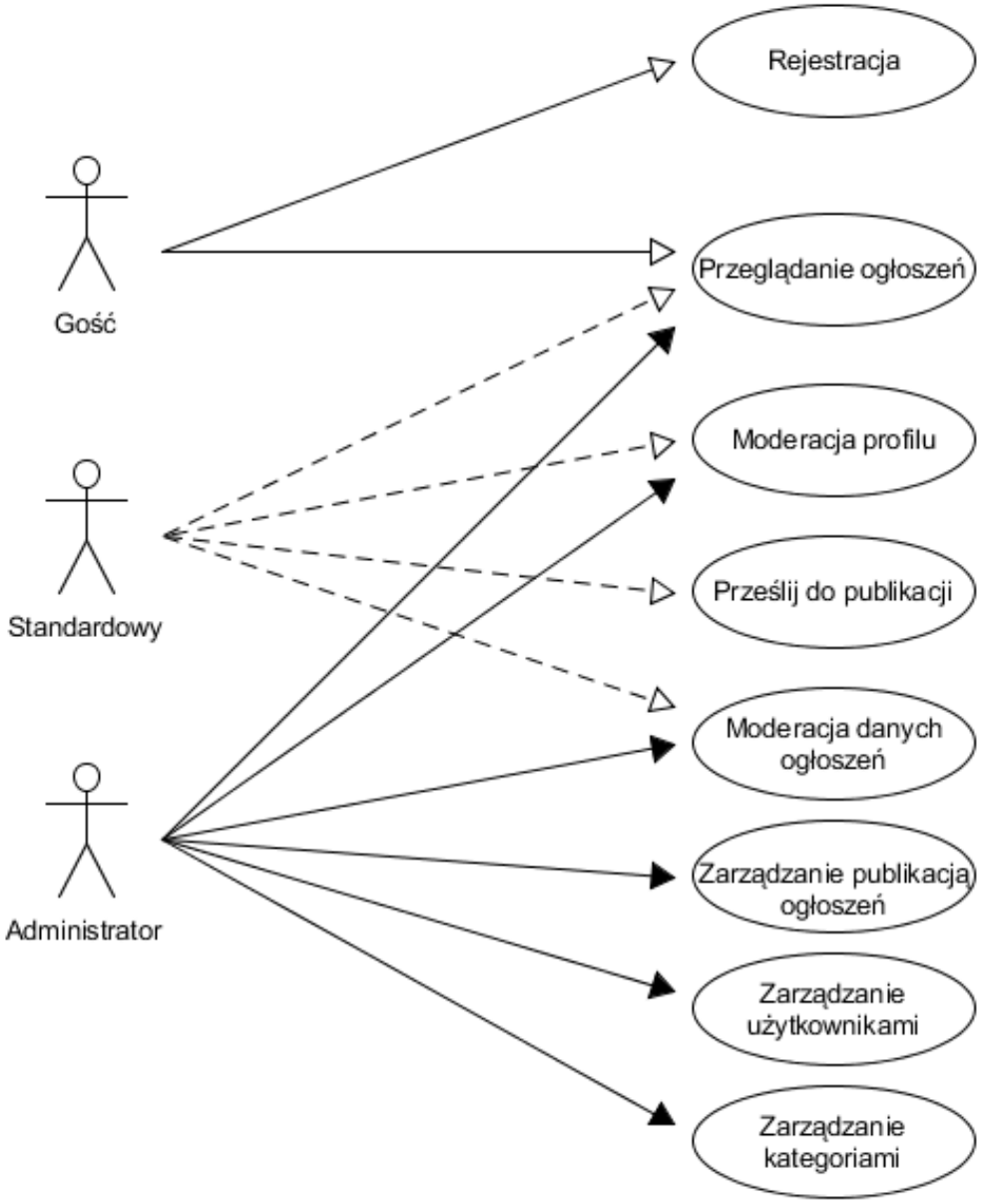
Wybranym przez nas hostingiem będzie **Heroku**. Jest to platforma chmurowa obsługująca kilka języków programowania w tym wybrany przez nas język Java.

Baza danych powinna zawierać tabele odpowiedzialne za użytkowników, ogłoszenia, kody pocztowe wraz ze współrzędnymi geograficznymi, kategorie oraz inne tabele niezbędne do przechowywania danych. Hasła oraz poufne dane powinny być szyfrowane za pomocą hashowania (BCrypt), którego użycie pozwala wcześniej wspomniane narzedzie Spring Security.

W celu poprawy bezpieczeństwa oraz poprawności danych zastosowana będzie walidacja danych zarówno we frontendzie jak i backendzie.
Aplikacja nie może mieć ograniczeń dotyczących ilości aktywnych użytkowników oraz publikowanych ogłoszeń. Maksymalny rozmiar portalu nie może przekraczać 1gb z powodu ograniczeń hostingowych.

Do przygotowania funkcji filtrowania należy zacząć od bazy danych kodów pocztowych oraz ich położenia geograficznego. Następnie napisać funkcję, która będzie szybko wyszukiwała wszystkie kody w wybranej przez użytkownika odległości.

3. Diagram UML



4. Schemat bazy danych

Advert:

Kolumna	Opis	Typ
Id	Identyfikator ogłoszenia	Int (Primary Key)
Title	Tytuł ogłoszenia	character varying (50), not null
Description	Opis ogłoszenia	character varying (1000)
Website	Strona domowa klubu	character varying (50)

Address	Adres klubu	character varying (200)
Email	Email klubu	character varying (50)
Phone	Telefon	character varying (30)
Status	Status ogłoszenia	character varying (30) , not null
Date	Data dodania ogłoszenia	date, not null
Postal_code	Kod pocztowy	character varying (10) , not null
Category_id	Klucz obcy	int
User_id	Klucz obcy	int

App_User:

Kolumna	**Opis**	**Typ**
Id	Identyfikator użytkownika	int , Primary Key
login	Nazwa użytkownika	character varying(30), not null
Password	Hasło użytkownika	character varying(100) , not null
Role	Rola	character varying(20) , not null
First_name	Imię	character varying(30),not null
Last_name	Nazwisko	character varying(30)
Email	Adres e-mail	character varying(30)

Category:

Kolumna	**Opis**	**Typ**
Id	Identyfikator kategorii	Int (Primary Key)
Name	Nazwa kategorii	Nchar(60), not null

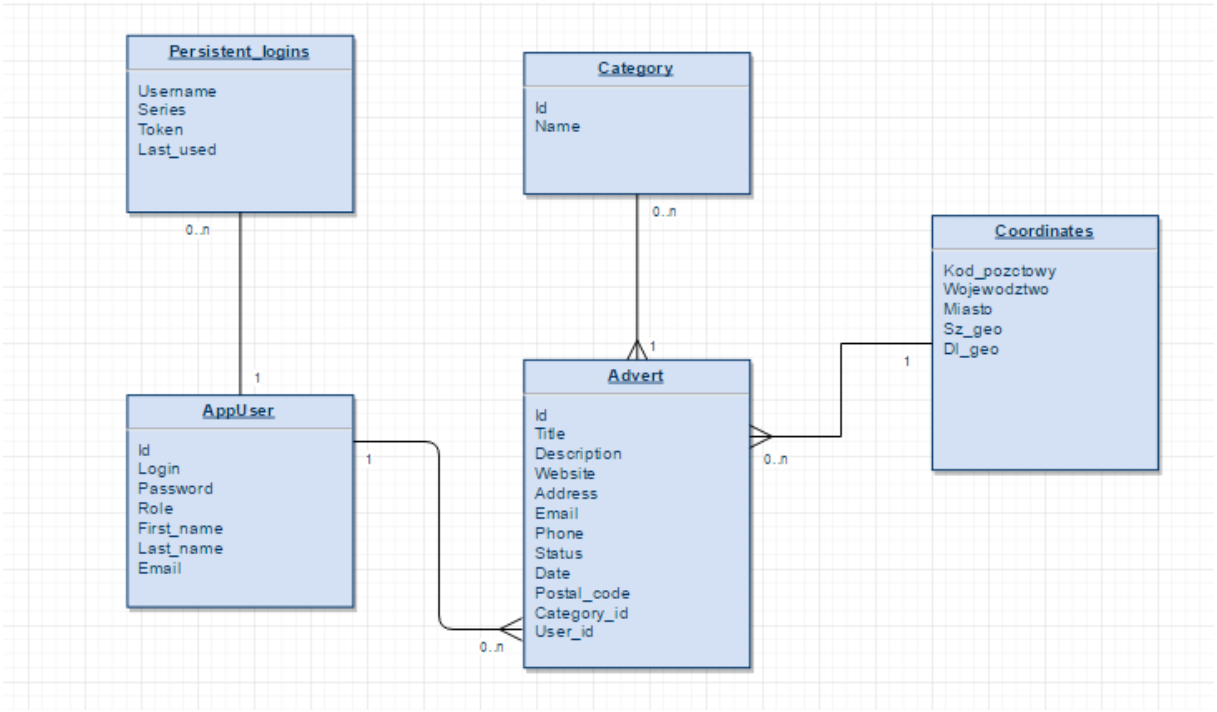
Coordinates:

Kolumna	**Opis**	**Typ**
Kod_pocztowy	Kod pocztowy	character varying, not null
wojewodztwo	Województwo	character varying(40)
Miasto	Miasto	character varying(40)
Sz_geo	Szerokość geograficzna	numeric
Dl_geo	Długość geograficzna	numeric

Persistent_logins

Kolumna	**Opis**	**Typ**
Username	Nazwa użytkownika	character varying(64)
Series		character varying(64)
Token		character varying(64)
Last_used	Ostatnie użycie	Timestamp

Diagram ERD:



Podczas uzupełnienia tabeli **Coordinates** niezbędne jest pobranie współrzędnych geograficznych miast wg kodów pocztowych. Pozwoli to na zastosowanie filtrowania klubów wg kodu pocztowego oraz wybranej przez nas odległości.

Aby pobrać dane zastosowano skrypt w języku PHP.

```
<?php

error_reporting(E_ALL & ~E_NOTICE);

set_time_limit(0);

require_once('ścieżka do biblioteki PHPExcel ');

$excelFile = "plik xls z kodami pocztowymi";

$pathInfo = pathinfo($excelFile);

$objReader = PHPExcel_IOFactory::createReader('Excel2007');

$objPHPExcel = $objReader->load($excelFile);

foreach ($objPHPExcel->getWorksheetIterator() as $worksheet) {

        $arkusz = $worksheet->toArray();
```

```
}

foreach ($arkusz as $row) {

	$gps = getCoordinates($row[0].' '.$row[1]);

	zapisanieDoPliku($row[0].';'.$row[1].';'.$row[2].';'.$gps);

}


function zapisanieDoPliku($dane){

$file = "kody.txt";

$fp = fopen($file, "a");

flock($fp, 2);

fwrite($fp, $dane."\r\n");

flock($fp, 3);

fclose($fp);

}


function getCoordinates($address){

$address = str_replace(" ", "+", $address);

$url = "http://maps.google.com/maps/api/geocode/json?sensor=false&address=$address";

$response = file_get_contents($url);

$json = json_decode($response,TRUE); //generate array object from the response from the web

return ($json['results'][0]['geometry']['location']['lat'].",".$json['results'][0]['geometry']['location']['lng']);

}

?>
```

Skrypt ten zapisuje współrzędne geograficzne do pliku txt, aby następnie dodać je do tabeli w bazie danych.

5. Repozytorium

Do utworzenia repozytorium zostanie wykorzystany serwis GitHub. Jest to system przeznaczony dla projektów programistycznych wykorzystujący system kontroli wersji Git.

Pozwala on na zarządzanie zadaniami, które są niezbędne do kontrolowania aplikacji podczas jej budowania.

6. Technologia programowania

Wybrana platforma programistyczna, w której będziemy tworzyć aplikację jest to Java EE. Stosuje się ją do tworzenia zarówno prostych stron www jak i po multiplatformowe aplikacje sieciowe. Standard ten jest tworzony przez firmę Oracle i dostarcza ona poza definicją interfejsów programistycznych, wzorcową implementację serwera aplikacyjnego.

7. Spring Security

Podczas tworzenia aplikacji został użyty Spring Security. Jest to narzędzie pozwalające na zabezpieczenie aplikacji.

Krok 1.

Aby dodać Spring Security do projektu należy do pliku pom.xml dodać bibliotekę org.springframework.security:

```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>${springsecurity.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>${springsecurity.version}</version>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>${springsecurity.version}</version>
</dependency>
```

Krok 2.

Pierwszą najważniejszą czynnością jest stworzenie pliku konfiguracyjnego Spring Security. Konfiguracja ta tworzy Servlet Filter, który będzie odpowiedzialny za bezpieczeństwo w naszej aplikacji. Przykładowo będzie odpowiedzialny za ochronę adresów URL, sprawdzanie poprawności użytkownika, przekierowanie do innych stron.

```
@Autowired
public void configureGlobalSecurity(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService);
    auth.authenticationProvider(authenticationProvider());
}
```

W metodzie powyżej ustawiamy w jaki sposób ma wyglądać zabezpieczenie. W naszym jako źródło będziemy używać:

```
@Bean
public DaoAuthenticationProvider authenticationProvider() {
    DaoAuthenticationProvider authenticationProvider = new DaoAuthenticationProvider();
    authenticationProvider.setUserDetailsService(userDetailsService);
    authenticationProvider.setPasswordEncoder(passwordEncoder());
    return authenticationProvider;
}
```

czyli metody, która dostarcza dane użytkownika i według tych danych będzie chroniła aplikację. Metoda ta przechowuje dane pobrane z bazy danych oraz dodatkowo szyfruje hasło BCryptPasswordEncoder'em.

Dodatkowo aby dodać do aplikacji funkcję zapamiętywania logowania użytkownika został użyty interfejs: PersistentTokenRepository, który przechowuje dodatkowo token wraz z jego datą dodania.

```
@Bean
public PersistentTokenBasedRememberMeServices getPersistentTokenBasedRememberMeServices() {
    return new PersistentTokenBasedRememberMeServices("remember-me", userDetailsService, tokenRepository);
}
```

Aby nasza aplikacja pamiętała, który użytkownik nie musi się logować, gdyż użył funkcji zapamiętywania logowania należy dodać nasze dane do bazy danych. Używamy do tego Hibernate i tworzymy model, który będzie zmapowany z tabelą PersistentLogin.

```
@Entity
@Table(name="PERSISTENT_LOGINS")
public class PersistentLogin implements Serializable{

    @Id
    private String series;

    @Column(name="USERNAME", unique=true, nullable=false)
    private String username;

    @Column(name="TOKEN", unique=true, nullable=false)
    private String token;

    @Temporal(TemporalType.TIMESTAMP)
    private Date last_used;
```

Następną czynnością jest utworzenie klasy CustomUserDetailsService, która zajmuje się przechowywaniem danych zalogowanego użytkownika oraz jego roli w aplikacji.

```
@Transactional(readOnly=true)
public UserDetails loadUserByUsername(String login)
    throws UsernameNotFoundException {
    User user = userService.findByLogin(login);
    logger.info("User : {}", user);
    if(user==null){
        logger.info("User not found");
        throw new UsernameNotFoundException("Username not found");
    }
    return new org.springframework.security.core.userdetails.User(user.getLogin(), user.getPassword(),
                                                                  true, true, true, true, getGrantedAuthorities(user));
}


private List<GrantedAuthority> getGrantedAuthorities(User user){
    List<GrantedAuthority> authorities = new ArrayList<>();

        authorities.add(new SimpleGrantedAuthority("ROLE_"+user.getRole()));
    logger.info("authorities : {}", authorities);
    return authorities;
}
```

Pozwala ona na dokonanie weryfikacji czy zalogowany użytkownik może dokonać czynności w aplikacji, które są niedostępne dla osoby niezalogowanej lub zwykłego użytkownika.

Kolejnym krokiem jest zarejestrowanie w aplikacji springSecurityFilter i robimy to poprzez następującą czynność.

```
public class SecurityWebApplicationInitializer extends AbstractSecurityWebApplicationInitializer {

}
```

Dodatkowo do ochrony aplikacji używamy token'ów. Pozwalają one na zabezpieczenie aplikacji przed atakami CSRF.

Do tego celu należy stworzyć klasę, w której możemy dostosować nasz serwer według naszych potrzeb.

```
@Override
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException, ServletException {

    HttpServletRequest request = (HttpServletRequest) req;
    HttpServletResponse response = (HttpServletResponse) res;

    response.setHeader("Access-Control-Allow-Origin", request.getHeader("Origin"));
    response.setHeader("Access-Control-Allow-Credentials", "true");
    response.setHeader("Access-Control-Allow-Methods", "POST, GET, OPTIONS, DELETE");
    response.setHeader("Access-Control-Max-Age", "3600");
    response.setHeader("Access-Control-Allow-Headers", "Origin, Content-Type, Accept, X-Requested-With, remember-me");
    response.setHeader("Vary", "Origin");
    response.setHeader("Access-Control-Allow-Origin", "https://znajdzklub.herokuapp.com");
    chain.doFilter(req, res);
}
```

Po utworzeniu i dodaniu tej klasy do konfiguracji Spring Security możemy używać tokenów w swojej aplikacji. Od tej pory, aby użyć np. metody POST do przesyłania danych, należy do żądania dodać odpowiedni token.

W tym celu do naszej strony w znaczniku head należy dodać nasz token.

```
<title>Znajdź klub</title>
<meta charset="utf-8">
<meta name="description" content="Znajdź klub - opis"/>
<meta name="viewport" content="width=device-width, initial-scale=1.0, minimum-scale=1.0"/>
<meta name="_csrf" content="${_csrf.token}"/>
<meta name="_csrf_header" content="${_csrf.headerName}"/>
<%@ include file="/app/basic/imports.jsp" %>
```

Następnie podczas wysyłania żądania przez klienta do serwera musimy do niego te wartości.

Jeśli wszystko przebiegnie poprawnie oraz token będzie prawidłowy to żądanie zostanie prawidłowo rozpatrzone, a jeśli nie to serwer nie wykona żadnej instrukcji i zwróci nam informację np. że nie można użyć metody POST w naszej aplikacji.

```
$http.defaults.headers.common[header] = token; - dodajemy tutaj token do
headera(request)

$http({
    method: 'POST',
    url: 'addCategory',
    data: dataObj,
    headers: {'Content-Type': 'application/json; charset=utf-8'}

}).success(function (data, status, headers, config) {
    if (data.success == true) {
        showAlert($scope, $mdDialog, 'Informacja', 'Poprawnie dodano
kategorię', function () {
            location.href = 'addCategory'
        });
    }
    else {
        showAlert($scope, $mdDialog, 'Błąd', data.error);
    }
}).error(function (data, status, headers, config) {
    showAlert($scope, $mdDialog, 'Błąd', "Błąd na serwerze");
}).finally(function () {
    $scope.disableMask();
});
```

Powyższa funkcja przesyła do url *addCategory* obiekt typu json. Jeśli serwer odpowie, że prawidłowo wysłane zostało żądanie to następnie zostanie sprawdzone czy zwrócony json nie zawiera błędów podczas dodawania.
Jeśli zawiera to na ekranie wyświetla się okienko z błędem, a jeśli nie to wyświetlana jest informacja, że poprawnie dodano kategorię i na końcu w metodzie finally wyłączana jest maska.

Ostatnim etapem jest dodanie do naszej konfiguracji Spring Security adresów url, które chcemy zabezpieczyć przed innymi osobami.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    http
        .addFilterBefore(new SimpleCORSFilter(), ChannelProcessingFilter.class)
        .authorizeRequests()
        .antMatchers("/userList").access("hasRole('ADMIN')")
        .antMatchers("/addCategory").access("hasRole('ADMIN')")
        .antMatchers("/edit-category-*").access("hasRole('ADMIN')")
        .antMatchers("/delete-category-*").access("hasRole('ADMIN')")
        .antMatchers("/addAdvert").access("isAuthenticated()")
        .antMatchers("/edit-advert-*").access("isAuthenticated()")
        .antMatchers("/delete-advert").access("isAuthenticated()")
        .antMatchers("/listUser").access("hasRole('ADMIN')")
        .antMatchers("/clubsList").access("hasRole('ADMIN')")
        .antMatchers("/user-*").access("hasRole('ADMIN')")
        .antMatchers("/edit-user-*").access("hasRole('ADMIN')")
        .antMatchers("/delete-user-*").access("hasRole('ADMIN')")
        .antMatchers("/userList").access("hasRole('ADMIN')").and().formLogin().loginPage("/login")
        .loginProcessingUrl("/login").usernameParameter("login").passwordParameter("password").and()
        .rememberMe().rememberMeParameter("remember-me").tokenRepository(tokenRepository)
        .tokenValiditySeconds(86400).and().csrf().and().exceptionHandling().accessDeniedPage("/Access_Denied");
}
```

Jak widać w konfiguracji jest już klasa z naszym filtrem CORS. Aby nasze adresy url były blokowane wg naszych wytycznych należy dodać .authorizeRequest() oraz dodać nasze adresy url.

Przykładowo adres „/edit-advert-*” jest dostępny tylko dla zalogowanych użytkowników. Niezalogowany użytkownik zostanie przekierowany na ekran logowania. Znak „*” oznacza, że oczekujemy na dowolny ciąg znaków., czyli jeśli adres zaczyna się od „edit-advert” to wykonuje się ona tylko dla osób zalogowanych.

W konfiguracji także widać metodę remember me oraz jej ważność.

W taki oto sposób możemy zabezpieczyć naszą aplikację przed niepowołanymi osobami