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.

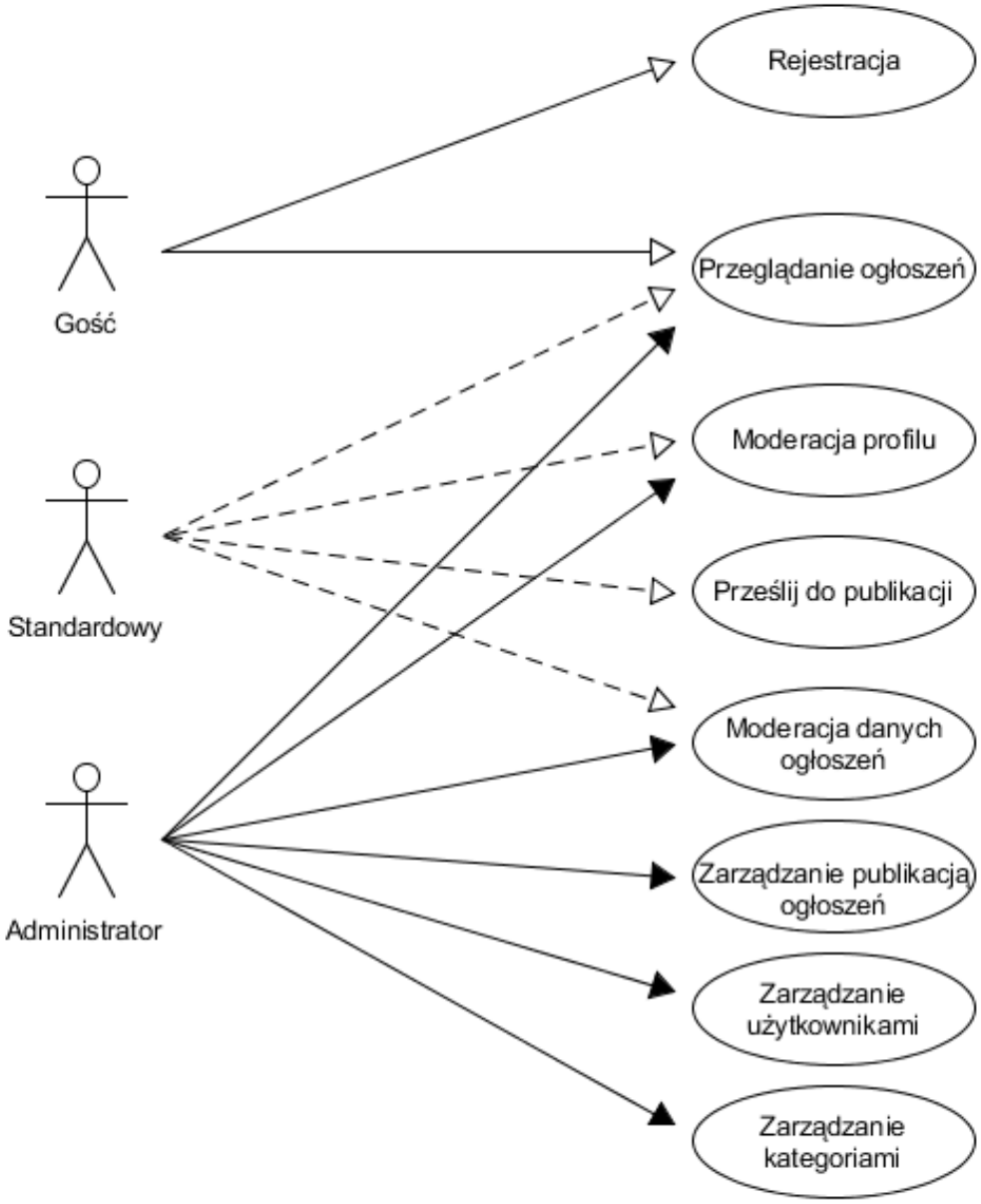
Wybranym przez nas hostingiem będzie **Heroku**. Jest to platforma chmurowa obsługująca kilka języków programowania w tym wybrany przez nas język Java.

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 (Aktualizowane)

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

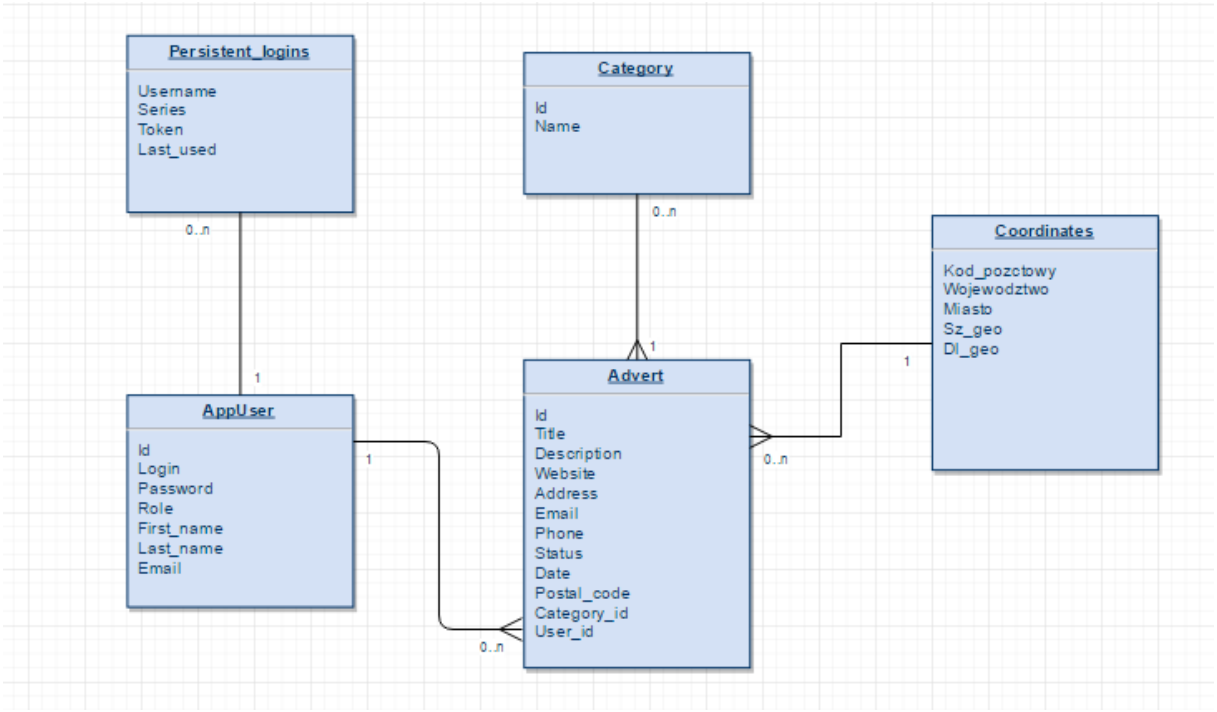
Coordinates:

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 = $objPHPExcel->getActiveSheet()->setCellValue('D'.$i, getCoordinates($row[0]+' '+$row[1]));

	$gps = getCoordinates($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 hostingowy 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. Omówienie wybranych metod z kontrolera

- pobieranie danych (json)

```
@RequestMapping(value = "/clubs-category-{id}", method =
RequestMethod.GET)
@ResponseBody
public List<Advert> getClubsByCategory(@PathVariable Integer id) {
    List<Advert> list = advertService.findByCategoryId(id);
    return list;
}
```

Adnotacja @RequestMapping pozwala na zmapowaniu metody do adresu url. W tym wypadku będziemy mogli wywołać metodę getClubsByCategory za pomocą adresu /clubs-category-{id}, gdzie id jest wartością wybranej kategorii. Ta metoda zwraca do naszej strony listę ogłoszeń z wybranej przez nas kategorii w formacie json przy pomocy funkcji GET.

- zapisywanie danych

```
@RequestMapping(value = {"/addAdvert"}, method = RequestMethod.POST,
produces = {"application/json; charset=UTF-8"})
@ResponseStatus(HttpStatus.OK)
public @ResponseBody ResponseEntity saveClub(@RequestBody @Valid Advert
advert, BindingResult result,
                                                ModelMap model) {
    JSONObject json = new JSONObject();
    if (!advertService.isUnique(advert)) {
        json.put("success", false);
        json.put("error", "Klub o podanym tytule już istnieje");
        return new ResponseEntity(json.toString(), HttpStatus.OK);
    }
    String userName = getPrincipal();
    User user = userService.findByLogin(userName);
```

```
    advert.setUserId(user.getId());
    advertService.save(advert);
    model.addAttribute("success", "Tytuł " + advert.getTitle() + " została
poprawnie dodana.");
    model.addAttribute("loggedinuser", userName);
    json.put("success", true);
    json.put("data", advert);
    return new ResponseEntity(json.toString(), HttpStatus.OK);
}
```

Tutaj jak w sytuacji powyżej mapujemy funkcję jako url - addAdvert. Tutaj widzimy, że funkcja ta czeka na wartości json przy pomocy POST.

W metodzie sprawdzamy czy tytuł ogłoszenia jest unikalny i jeśli tak to jest ono dodawane do bazy danych a metoda zwraca komunikat o prawidłowym dodaniu ogłoszenia.

- bezpieczeństwo

```
@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");
}
```

W celu zabezpieczenia aplikacji internetowej Spring Security pozwala na dodanie adresów url, do których określamy dostęp. Jak widać do adresu *userList* dostęp ma tylko użytkownik z uprawnieniem ADMIN. Następnie z url *addAdvert* może korzystać każdy zalogowany użytkownik.

Jeśli zalogowany użytkownik bez odpowiedniej roli będzie próbował użyć nieprzydzielonych mu opcji wyświetli się strona mówiąca o braku uprawnień użytkownika, a dla niezalogowanego użytkownika będzie wyświetlała się strona logowania.

Kolejnym aspektem bezpieczeństwa jest zastosowanie tokenów. Podczas akcji POST bez odpowiedniego tokena otrzymamy komunikat o braku możliwości używania tej metody. Tworzymy go w znaczniku *head* a pobieramy go podczas przesyłania danych w formularzu.

```
Formularz:
<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
```

Kontroler js:

```
$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.

8. Widok

```
<div class="form-group" ng-class="{ 'has-error' : form.name.$invalid
&& !form.name.$pristine }">
    <label>Nazwa *</label>
    <input type="text" name="name" class="form-control" ng-
model="category.name" required>
    <p ng-show="form.name.$invalid && !form.name.$pristine"
class="help-block">Pole nazwa jest obowiązkowe.</p>
</div>
```

W formularzach została zastosowana walidacja. W powyższym przykładzie widzimy walidację front-endową pola Nazwa.

Sprawdzamy tu czy znacznik input z name=”name” jest pusty. Jeśli jest pusty to na ekranie pojawia się komunikat „Pole nazwa jest obowiązkowe”. Jeśli wszystkie pola są prawidłowo wypełnione to na ekranie zostaje aktywowany przycisk ‘Zarejestruj’.

9. Walidacja

W aplikacji została użyta walidacja front-endowa, co zostało pokazane we wcześniejszym przykładzie oraz podczas rejestracji back-end'owa.

Przykład:

```
private String waliduj(User user) {
    String error = "";
    Pattern pemail = Pattern.compile(".+@.+");
    Matcher memail = pemail.matcher(user.getEmail());
    if (!memail.matches()) {
        error += "adres email jest nieprawidłowy.\n";
    }

    if (user.getLogin().length() < 3) {
        error += "login jest za krótki.\n";
    } else if (user.getLogin().length() > 30){
        error += "login jest zbyt długi";
    }

    if (user.getPassword().length() < 5) {
        error += "hasło jest zbyt krótkie \n";
    }
    return error;
}
```

Jeśli walidacja wykryje niepoprawność danych to zostaną one przesłane do klienta w postaci jsona i zostaną one wyświetlone na okienku.

10. Model

```java
@Entity
@Table(name = "CATEGORY")
public class Category implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "CATEGORY_ID")
    private Integer id;

    @NotEmpty
    @Column(name = "NAME", unique = true, nullable = false)
    private String name;

    public Integer getId() { return id; }

    public void setId(Integer id) { this.id = id; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }
```

Powyżej został pokazany model kategorii. Widzimy, że model ten odnosi się do tabeli w bazie danych o nazwie *Category*. Obiekt Category zawiera id kategorii oraz nazwę. Widzimy, że kolumna name zwiera adnotację @NotEmpty czyli pole w obiekcie nie może być puste.