

**Dokumentacja projektowa**  
aplikacji webowej:  
portal ogłoszeń motoryzacyjnych

Mateusz Czerski   Valeriia Ząbkowska   Aleksandra Bąk

## **Spis treści**

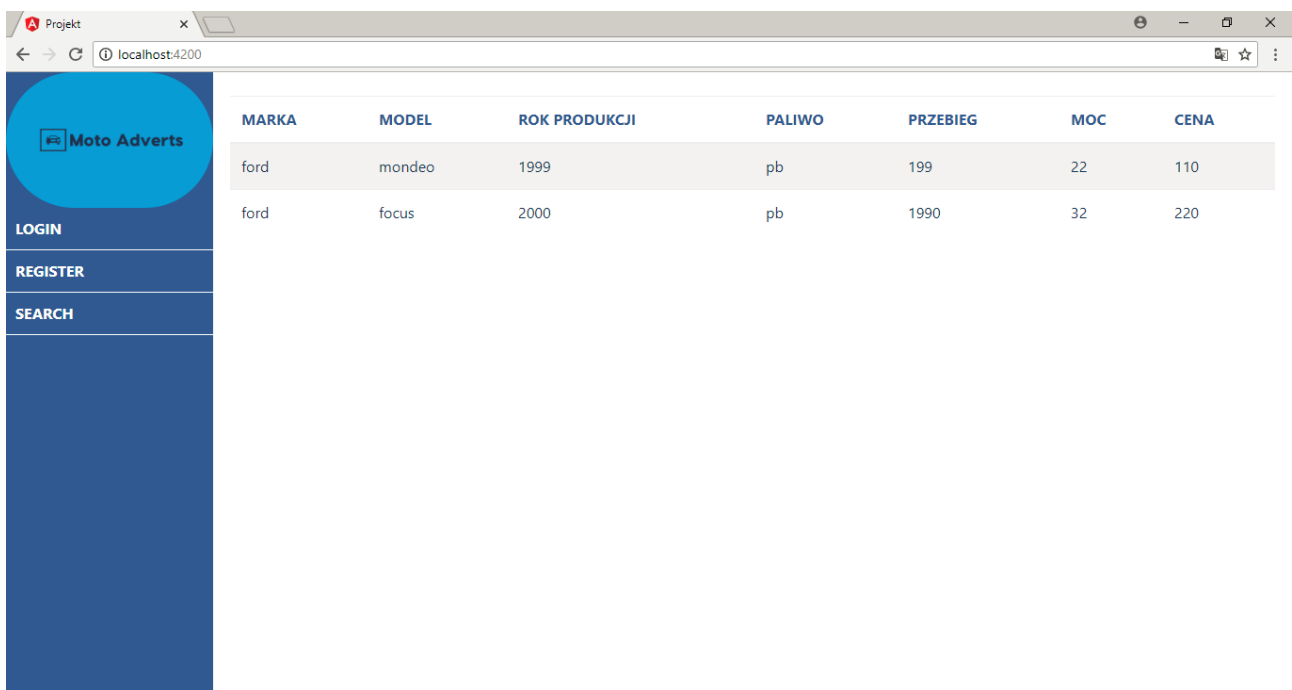
1. Opis projektu
2. Założenia funkcjonalne i нефункционалне
3. Zespół
4. Technologie
5. Projekt systemu
  - 5.1. Architektura systemu
  - 5.2. Logika aplikacyjna
6. Podsumowanie

Dokumentacja ta została stworzona przy tworzeniu aplikacji webowej, która jest portalem internetowym służącym do przeglądania oraz dodawania ogłoszeń motoryzacyjnych. W skład niniejszej dokumentacji wchodzi:

- opis założeń funkcjonalnych i нефункциональных projektu,
- opis projektu pod względem wykorzystanej architektury oraz logiki aplikacji.

## 1.Opis projektu

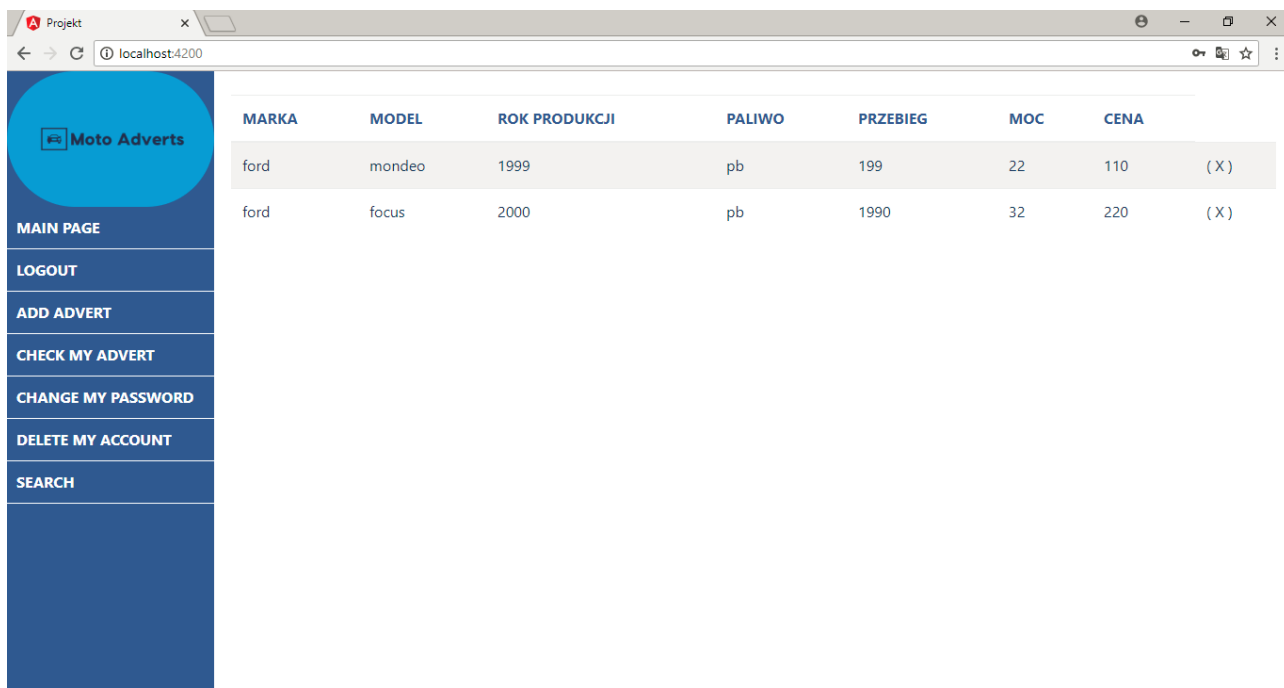
Projekt zawarty w dokumentacji projektowej dotyczy portalu ogłoszeń motoryzacyjnych. Do podstawowych zadań portalu należą przede wszystkim: przeglądanie i wyszukiwanie ogłoszeń motoryzacyjnych, a także ich dodawanie po zalogowaniu się do panelu użytkownika serwisu.



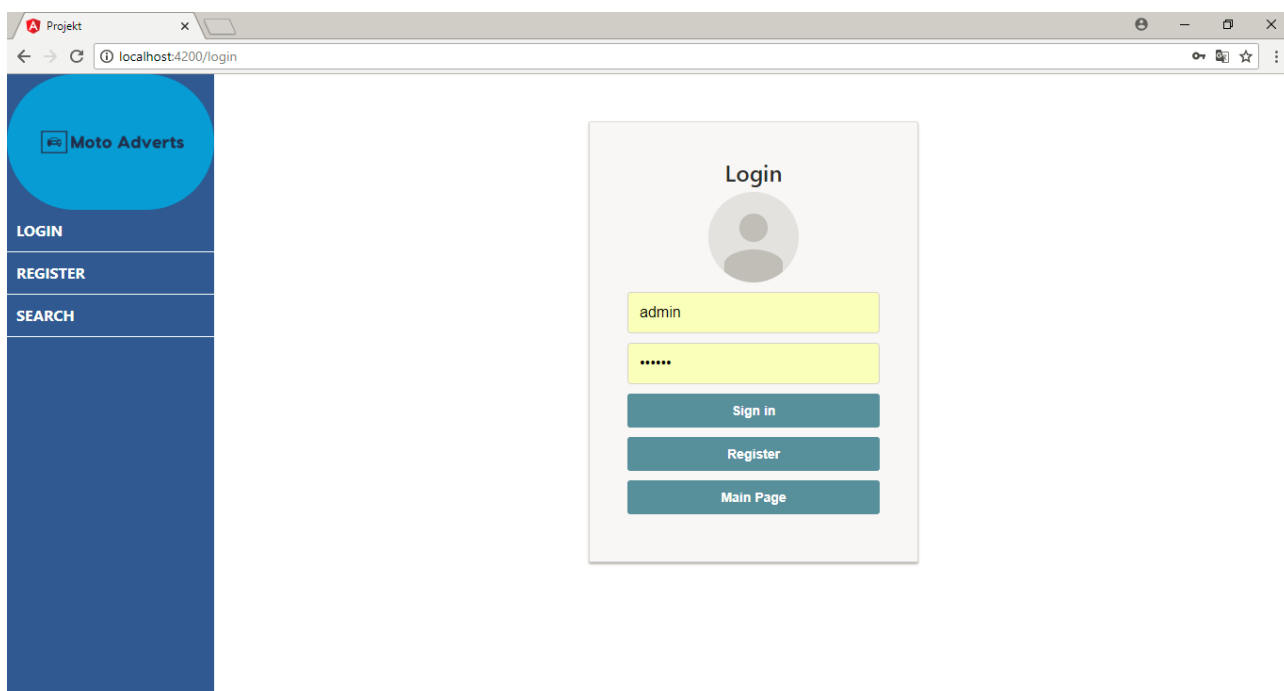
MARKA	MODEL	ROK PRODUKCJI	PALIWO	PRZEBIEG	MOC	CENA
ford	mondeo	1999	pb	199	22	110
ford	focus	2000	pb	1990	32	220

Rys. Strona główna aplikacji (widoczna dla wszystkich)

Oprócz wcześniej wymienionych zadań, użytkownik ma również możliwość rejestracji konta w serwisie, dodawania, modyfikacji i usuwania ogłoszeń, a także dodawania, modyfikacji i usuwania swojego konta. Więcej funkcjonalności zostało opisanych w rozdziale drugim (patrz rozdz. 2. Założenia funkcjonalne i нефункциональные).



Rys. Strona główna aplikacji (widoczna dla dla zalogowanego użytkownika)



Rys. Widok logowania

**Registration**

Username

Enter your Username

Password

Enter your Password

Confirm Password

Confirm your Password

Register

Login

Main Page

Rys. Widok rejestracji

**SEARCH**

MARKA

MODEL

PALIWO

ROK PRODUKCJI - OD

ROK PRODUKCJI - DO

PRZEBIEG - OD

PRZEBIEG - DO

MOC - OD

MOC - DO

CENA - OD

CENA - DO

Search Reset

MARKA	MODEL	ROK PRODUKCJI	PALIWO	PRZEBIEG	MOC	CENA	
ford	mondeo	1999	pb	199	22	110	(X)
ford	focus	2000	pb	1990	32	220	(X)

Rys. Strona wyszukiwania ogłoszeń (widok zalogowanego użytkownika)

Dodatkowo do serwisu został zaprojektowany panel administratora, poprzez który administrator może wprowadzać zmiany dotyczące działania aplikacji. W panelu tym możliwe do wykonania akcje, to m. in. usuwanie i modyfikowanie kont oraz ogłoszeń użytkownika.

## **2. Założenia funkcjonalne i нефункционалне**

Celem projektu jest stworzenie serwisu online z ogłoszeniami motoryzacyjnymi. Poniżej zostały zestawione ogólne założenia funkcjonalne i нефункционалне aplikacji oraz funkcje programu dostępne dla poszczególnych grup użytkowników – niezalogowanych (gości), zalogowanych oraz administratora.

### **Założenia funkcjonalne:**

W systemie przewidziane są 2 role: użytkownik i administrator.

Funkcje dostępne dla administratora:

- dostęp do panelu administracyjnego,
- usuwanie kont użytkowników.
- usuwanie i modyfikacja ogłoszeń dodanych przez użytkowników.
- przeglądanie listy ogłoszeń motoryzacyjnych i ich szczegółów,
- przeszukiwanie ogłoszeń (parametryzowane).

Funkcje dostępne dla użytkownika niezalogowanego(gościa):

- tworzenie konta użytkownika wraz z jego weryfikacją via email,
- logowanie do systemu (podanie loginu/email i hasła),
- przeglądanie listy ogłoszeń obecnych w bazie i ich szczegółów,
- przeszukiwanie ogłoszeń (parametryzowane).

Funkcje dostępne dla użytkownika zalogowanego:

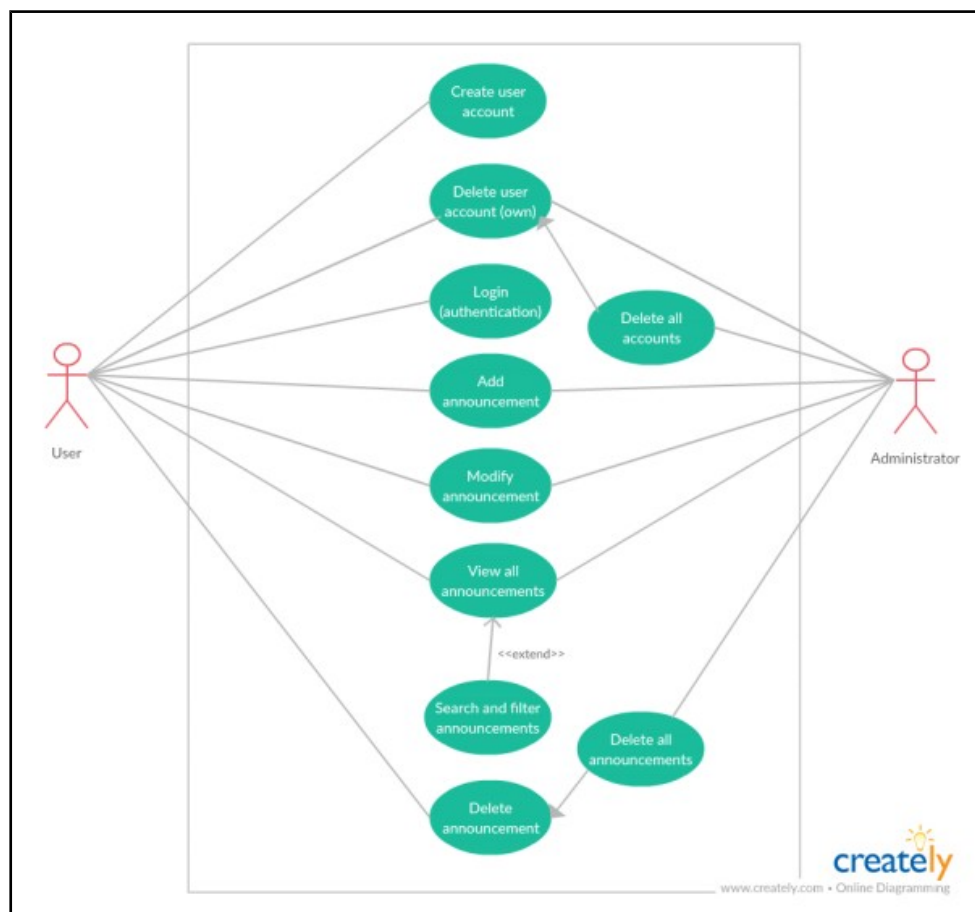
- przeglądanie listy ogłoszeń obecnych w bazie i ich szczegółów,
- przeszukiwanie ogłoszeń (parametryzowane),
- dostęp do panelu użytkownika,
- dodawanie własnych ogłoszeń,
- modyfikacja własnych ogłoszeń,
- usuwanie własnych ogłoszeń,

- usuwanie własnego konta,
- wylogowanie z aplikacji.

### Założenia niefunkcjonalne:

- oprogramowanie jest responsywne i automatycznie dostosowuje się do różnych rozdzielczości ekranu;
- aplikacja ma działać oraz wyglądać podobnie w najpopularniejszych przeglądarkach.

Diagram przypadków użycia:



Na powyższym diagramie przypadków użycia dotyczącego możliwych do wykonania akcji w portalu ogłoszeń motoryzacyjnych, widoczni są dwaj aktorzy, którymi są Użytkownik oraz Administrator serwisu. Użytkownik niezalogowany (gość) ma do wyboru akcje dotyczące przeglądania, wyszukiwania, filtrowania wszystkich ogłoszeń oraz rejestracji konta użytkownika. Użytkownik po poprawnej rejestracji i zalogowaniu może dodatkowo dokonywać akcje związane z dodawaniem, modyfikacją i usuwaniem ogłoszenia, a także usuwaniem konta. Administrator może dokonać akcji związanych z usuwaniem konta użytkownika oraz ogłoszeń, a także może podejmować akcje związane z dodawaniem, modyfikowaniem i wyświetlaniem ogłoszeń.

### **3.Zespół**

- Mateusz Czerski
- Valeriia Ząbkowska
- Aleksandra Bąk

### **4. Technologie**

- Angular4
- JavaScript
- Node.js
- HTML
- CSS
- JAVA
- Maven
- SQL
- MySQL
- Hibernate
- JPA
- cors
- JWT



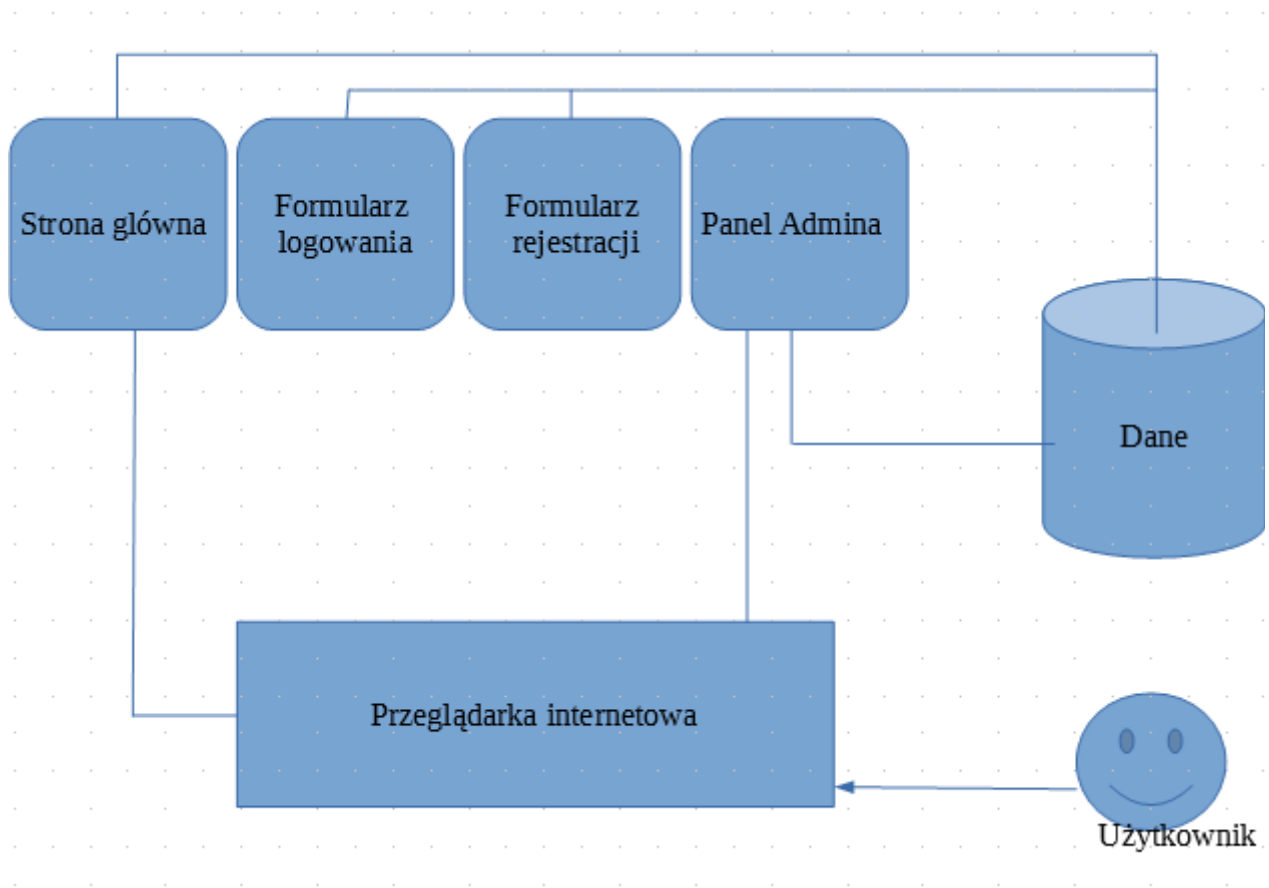
## 5. Projekt systemu

### 5.1. Architektura systemu

Panel administracyjny dostępny tylko dla użytkownika z uprawnieniami administratora. Panel ten pozwala na zarządzanie bazą ogłoszeń oraz użytkownikami. Operacje wykonywać można przy pomocy tabel.

Strona główna aplikacji umożliwia użytkownikom przeglądanie oraz wyszukiwanie ogłoszeń.

Panel użytkownika zalogowanego umożliwia poprzez formularz dodanie własnych ogłoszeń, zarządzanie swoimi ogłoszeniami oraz kontem. Poniżej została rozpisana logiczna architektura aplikacji.



Rys. Architektura logiczna aplikacji

Frontowa część projektu została utworzona w oparciu o framework Angular4, którego głównym założeniem jest tworzenie aplikacji internetowych typu Single Page Application. Platforma korzysta z języka TypeScript. Do stworzenia całej struktury aplikacji został użyty język HTML. Za wygląd oprogramowania odpowiada arkusz stylów CSS oraz framework CSS - Bootstrap.

Aplikacja korzysta również z pakietów dostępnych dla frameworka Angular4. Najważniejsze z nich: core, js, rxjs, ts-node, tslint.

Dla uruchomienia frontowej części aplikacji w źródłowym folderze ze okna poleceń: npm install – komenda do instalowania niezbędnych pakietów oraz npm start – uruchomienie aplikacji.

W przeglądarce pod adresem <http://localhost:4200/> będzie strona główna aplikacji.

Baza danych od strony serwera zarządzana jest przez relacyjny system zarządzana MySQL.

## 5.2. Logika aplikacyjna

### *Komponenty aplikacji*

Podstawowym składnikiem Angular4 służącym do wyświetlania treści w aplikacji są komponenty. Każdy komponent posiada widok umieszczony w odpowiednim HTML pliku.

Komponent kontroluje fragment ekranu - widok. Widoki aplikacji są kontrolowane przez komponenty:

MainPageComponent – zawiera stronę główną aplikacji,

LoginComponent - zawiera formularz logowania użytkownika do serwisu na wcześniej utworzone konto,

RegisterComponent – zawiera formularz rejestracji nowego użytkownika,

AddCarComponent – zawiera formularz dodawania nowego auta do bazy danych,

MyAdvertComponent - zawiera listę ogłoszeń użytkownika,

CarDetailsComponent – zawiera szczegóły wybranego ogłoszenia,

ChangePassComponent - zawiera funkcje zmiany hasła użytkownikiem

CheckAccountsComponent - zawiera funkcje zarządzania kontem użytkownika

SearchComponent – zawiera formularz wyszukiwania w bazie ogłoszeń.

Wewnątrz klasy komponentu zdefiniowana logika aplikacji, tzn. jak komponent obsługuje widok.

Przykład komponentu MyAdvertComponent, który obsługuje widok ogłoszeń konkretnego użytkownika z możliwością usunięcia poszczególnych ogłoszeń (metoda removedCar):

```
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';
import { CarService } from '../car-table-row/car.service';
import { Car } from '../model';

@Component({
  selector: 'app-my-advert',
  templateUrl: './my-advert.component.html',
  styleUrls: ['./my-advert.component.less']
})
export class MyAdvertComponent implements OnInit {

  constructor(private carService: CarService, private router: Router) { }

  cars: Car[];
  ngOnInit() {
    this.loadMyCars();
  }

  loadMyCars(): void {
    this.carService.getMyCars().subscribe((cars) => {
      this.cars = cars;
    });
  }

  removedCar(car: Car) {
    this.carService.removeCar(car.id).subscribe(() => {
      this.loadMyCars();
    });
  }
}
```

## Module

Moduły Angular - klasa utworzona/zmodyfikowana dekoratorem `@NgModule` - są podstawową cechą Angulara. JavaScript (TypeScript) ma swój własny system modułów do zarządzania kolekcjami obiektów JavaScript. To jest kompletnie inny i niezwiązany z Angulariem system modułów. W JavaScript każdy plik jest modułem i wszystkie obiekty zdefiniowane w tym pliku należą do tego modułu. Moduł deklaruje niektóre obiekty jako publiczne oznaczając je słowem kluczowym `export`. Inne moduły JavaScript używają polecenia `import` aby dostać się do publicznych obiektów innych modułów. To są dwa inne ale komplementarne systemy modułów. Obu użyte zostały do tworzenia aplikacji.

Przykład modułu `cars.module`:

```
import { CarTableRowComponent } from './car-table-row/car-table-row.component';
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { RouterModule } from '@angular/router';

@NgModule({
  imports: [
    CommonModule,
    RouterModule
  ],
  exports: [CarTableRowComponent],
  declarations: [CarTableRowComponent]
})
export class CarsModule {}
```

## Metadane

Metadane określają w jaki sposób Angular4 ma przetwarzać klasy. W TypeScript dekorator `@Component` określa, że klasa występująca bezpośrednio za nim jest klasą komponentu. Metadane dla komponentu `AddCar`:

```
@Component({
  selector: 'app-add-car',
  templateUrl: './add-car.component.html',
  styleUrls: ['./add-car.component.less']
})
```

## Szablony

Widok związany z komponentem z użyciem towarzyszącego mu szablonu. Szablon jest formą HTMLa, która informuje Angular4 jak wyświetlić komponent. Szablony wyglądają jak zwyczajne HTML, z kilkoma różnicami. Kod taki jak ngModel, (click) używa składni szablonu Angular4.

Przykład Szablonu dla widoku logowania:

```
<div class="container">
  <div class="card card-container">
    <h4 class="text-center">Login</h4>
    
    <form (submit)="onSubmit()">
      <div class="form-signin">
        <input [(ngModel)]="user.username" name="login" type="text" id="inputEmail"
          | class="form-control" placeholder="Login">
        <input [(ngModel)]="user.password" name="password" type="password" id="inputPassword"
          | class="form-control" placeholder="Password">
        <button class="btn btn-lg btn-primary btn-block btn-signin" type="submit">
          Sign in</button>
        <button (click)="ReDirect('register')" class="btn btn-lg btn-primary btn-block btn-signin"
          | type="submit">Register</button>
        <button (click)="ReDirect('')" class="btn btn-lg btn-primary btn-block btn-signin"
          | type="submit">
          Main Page</button>
      </div>
    </form>
  </div>
</div>
```

## Routing

Jednym z najważniejszych składników aplikacji jest routing. Umożliwia nawigowanie z jednego widoku do drugiego jako konsekwencja podejmowanych przez użytkownika aktywności. W ramach tej aplikacji do obsługi routingu został zastosowany, pakiet angular/router.

Korzystając z tego pakietu możemy również ustalić dostęp wybranych użytkowników do poszczególnych części aplikacji. W przypadku tego oprogramowani ważne jest, aby dostęp do panelu administracyjnego posiadał tylko użytkownik z uprawnieniami administratora.

Dostępne routes do nawigacji:

```
const APP_ROUTES: Route[] = [  
  { path: '', component: MainPageComponent },  
  { path: 'login', component: LoginComponent },  
  { path: 'register', component: RegisterComponent },  
  { path: 'addCar', component: AddCarComponent },  
  { path: 'myAdvert', component: MyAdvertComponent },  
  { path: 'cars/:id', component: CarDetailsComponent},  
  { path: 'pass', component: ChangePassComponent},  
  { path: 'accounts', component: CheckAccountsComponent},  
  { path: 'search', component: SearchComponent}  
];
```

## Serwisy

Serwis w Angularze4 to szeroka kategoria obejmująca dowolną wartość, funkcję czy cechę, której potrzebuje nasza aplikacja. Komponenty są odbiorcami tych usług. Korzystamy z 3 obiektów usług, a mianowicie: `LoginService`, `CarService`, `LayoutService`.

`LoginService` posiada metody `AddUser`, `DeleteAcc`, `register`, `ChangePass`, `getAccounts`, `removeAcc`. Przykładowo, metoda `AddUser` dodaje do bazy nowego użytkownika, a metoda `DeleteAcc` kasuje jego z bazy.

```
AddUser(user) {  
  let headers = new Headers({ 'Content-Type': 'application/json' });  
  let options = new RequestOptions({ headers: headers });  
  let body = JSON.stringify(user);  
  return this.http.post('http://localhost:8080/demo/login', body, options)  
    .map(res => res.json());  
}  
  
DeleteAcc() {  
  let myHeaders = new Headers();  
  myHeaders.append('Authorization', localStorage.getItem('token').replace('\"', '').replace('\"', ''));  
  myHeaders.append('Content-Type', 'application/json');  
  let options = new RequestOptions({ headers: myHeaders });  
  return this.http.delete('http://localhost:8080/demo/deleteAccount', options);  
}
```

`CarService` nawiązuje połączenie przez serwer z bazą danych dla uzyskania listy istniejących ogłoszeń, poszczególnych ogłoszeń, dodawania nowych oraz usunięcia istniejących ogłoszeń. Przykładowe metody `updateCar` oraz `removeCar`.

```
updateCar(id: number, data): Observable<any> {  
  let myHeaders = new HttpHeaders();  
  myHeaders = myHeaders.append('Authorization', localStorage.getItem('token').replace('\"', '').replace('\"', ''));  
  myHeaders = myHeaders.append('Content-Type', 'application/json');  
  console.log(data);  
  return this.http.post('http://localhost:8080/demo/updateCar' + `/${id}`, data, { headers: myHeaders });  
}  
  
removeCar(id: number): Observable<any> {  
  let myHeaders = new HttpHeaders();  
  myHeaders = myHeaders.append('Authorization', localStorage.getItem('token').replace('\"', '').replace('\"', ''));  
  myHeaders = myHeaders.append('Content-Type', 'application/json');  
  return this.http.delete('http://localhost:8080/demo/deleteCar' + `/${id}`, { headers: myHeaders });  
}
```



*LayoutService* ustawia jaki widok Sidebar będzie wyświetlany w przeglądarce w zależności od tego, jaki użytkownik korzysta z aplikacji – gość, zalogowany użytkownik lub admin.

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs/BehaviorSubject';

@Injectable()
export class LayoutService {

  sidebarSource$ = new BehaviorSubject<number>(0);
  showSidebar() {
    if (localStorage.getItem('admin')) {
      this.sidebarSource$.next(2);
    } else {
      this.sidebarSource$.next(1);
    }
  }
  hideSidebar() {
    this.sidebarSource$.next(0);
  }
}
```

## ***Projekt serwera oraz warstwy bazy danych***

Backend jest połączeniem między frontem i bazą danych, wykonuje on operacje typu update, insert, delete. Operacje tego typu jak i wyciąganie danych jest możliwe za pomocą usług tzw. restów. Backend dodatkowo posiada metody zabezpieczające przed kradzieżą hasła i włamaniem do aplikacji. Wykorzystuje on nowoczesny algorytm szyfrowania w technologii jwt.

Przykład klasy JWTAuthenticationFilter wraz z importami.

```
package restApi.security;

import java.io.IOException;
import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Component;
import org.springframework.web.context.support.SpringBeanAutowiringSupport;
import org.springframework.web.filter.GenericFilterBean;

@Component
public class JWTAuthenticationFilter extends GenericFilterBean {
    @Autowired
    TokenAuthenticationService token;

    @Override
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)
        throws IOException, ServletException {
        SpringBeanAutowiringSupport.processInjectionBasedOnCurrentContext(this);

        Authentication authentication = token.getAuthentication((HttpServletRequest) request);

        SecurityContextHolder.getContext().setAuthentication(authentication);
        filterChain.doFilter(request, response);
    }
}
```

## Cors

Dodatkowo do połączenia z częścią frontendowa ustawiamy corsy dzięki którym serwer backend rozpoznaje aplikacje frontend i umożliwia jej autoryzację i uwierzytelnienie.

```
public class MyCors extends CorsConfiguration{
    private List<String> allowedOrigins;
    private List<String> allowedMethods;
    private List<HttpMethod> resolvedMethods = DEFAULT_METHODS;
    private List<String> allowedHeaders;
    private List<String> exposedHeaders;
    private Boolean allowCredentials;
    private Long maxAge;
    public MyCors() {}
    private static final List<HttpMethod> DEFAULT_METHODS;
    static {
        List<HttpMethod> rawMethods = new ArrayList<HttpMethod>(2);
        rawMethods.add(HttpMethod.GET);
        rawMethods.add(HttpMethod.HEAD);
        DEFAULT_METHODS = Collections.unmodifiableList(rawMethods);
    }
    @Override
    public CorsConfiguration applyPermitDefaultValues() {
        if (this.allowedOrigins == null) {
            this.addAllowedOrigin(ALL); }
        if (this.allowedMethods == null) {
            this.setAllowedMethods(Arrays.asList(
                HttpMethod.GET.name(), HttpMethod.HEAD.name(), HttpMethod.POST.name(), HttpMethod.DELETE.name())); }
        if (this.allowedHeaders == null) {
            this.addAllowedHeader(ALL); }
        if (this.allowCredentials == null) {
            this.setAllowCredentials(true); }
        if (this.maxAge == null) {
            this.setMaxAge(1800L); }
        return this;
    }
}
```

Warstwa danych oprogramowania wspierana jest przez system zarządzania bazą danych MySQL.

## 6. Podsumowanie

W projekcie została zaimplementowana aplikacja webowa dotycząca portalu ogłoszeń motoryzacyjnych. Cała implementacja oprogramowania została stworzona zgodnie z przyjętymi na początku założeniami funkcjonalnymi i нефункциональными.

Główną funkcją aplikacji jest możliwość przeglądania przez gościa serwisu wszystkich ogłoszeń motoryzacyjnych zawartych w serwisie. Przeglądanie to usprawnia filtrowanie wyników, którego rezultat jest zgodny z parametrami wybranymi przez użytkownika. Użytkownik może również dokonać rejestracji na portalu, dzięki czemu zyskuje dostęp do korzystania z pełnej funkcjonalności zaprojektowanej po jego stronie. Do funkcjonalności tej należą dodawanie, modyfikowanie i usuwanie ogłoszenia, a także zarządzanie swoim kontem oraz jego usuwanie.

Kolejną szczególnie przydatną funkcją serwisu jest stworzony panel administracyjny, w którym właściciel serwisu może nim zarządzać. Panel ten umożliwia administratorowi modyfikowanie, przeglądanie i usuwanie ogłoszeń oraz modyfikowanie i usuwanie kont zarejestrowanych użytkowników.

Strony aplikacji działają zgodnie z międzynarodowymi standardami sieciowymi W3C. Ponadto prawidłowo wyświetlają się we wszystkich nowych wersjach popularnych przeglądarek internetowych oraz na urządzeniach o średniej i dużej rozdzielczości ekranu.