



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
Wydział Inżynierii Metali i Informatyki Przemysłowej

Portal motoryzacyjny

Autor:	Kacper Bielak
Kierunek studiów:	Informatyka techniczna
Nr albumu:	400143
Przedmiot:	Zawansowane techniki internetowe

Kraków, 27.05.2024r.

Spis treści

1. Wstęp i cel projektu	2
2. Funkcjonalności zaimplementowanego systemu	3
3. Technologie wykorzystane do implementacji portalu	4
4. Implementacja systemu.....	6
4.1. Implementacja systemu bazodanowego wykorzystane w systemie.....	6
4.2. Implementacja backendu	9
4.3. Implementacja frontendu	13
5. Prezentacja stworzonego systemu.....	15
6. Wnioski i możliwości dalszego rozwoju	23

1. Wstęp i cel projektu

Celem projektu było stworzenie aplikacji webowej będącej portalem motoryzacyjnym, która ma w przyszłości zastąpić istniejące na portalu Facebook grupy przeznaczone do wymieniać się informacjami na temat konkretnych pojazdów między użytkownikami. Podjęto próbę implementacji takiego systemu ze względu na słabą przejrzystość oraz przerost informacji z różnych dziedzin na ww. portalu. Ponadto dla każdego modelu pojazdu, nie samej marki, istnieje oddzielna grupa przez co ciężko jest wylapać informację na temat konkretnego modelu. Co więcej, taka forma nie zapewnia możliwości wyszukania najczęściej pojawiających się usterek zgłoszonych przez posiadaczy pojazdów co wydaje się być bardzo przydatne np. w przypadku chęci zakupu danego modelu.

Z powyższych względów stworzono system umożliwiający wymianę informacji między użytkownikami ze sztywno przypisanymi markami i opcjonalnie modelami pojazdów. Portal umożliwia przeglądanie postów użytkownikom niezalogowanym wraz z ustawianiem odpowiednich filtrów takich jak marka i model pojazdu, najczęściej pojawiające się usterki czy typ postu, a także samo sortowanie wyników. System umożliwia rejestrację kont dzięki odpowiedniemu panelowi oraz zmianę danych dla już istniejących użytkowników. Użytkownicy zalogowani mogą natomiast dodatkowo dodawać / usuwać / edytować swoje własne posty, komentować istniejące czy je podbijać. Portal obsługuje również dodawanie zdjęć do wpisów i komentarzy, a także posiada moduł powiadomień dzięki czemu żaden komentarz nie umknie osobie tworzącej wątek.

2. Funkcjonalności zaimplementowanego systemu

Przed implementacją systemu postawiono szereg wymagań funkcjonalnych przed projektem. Poniżej przedstawiono funkcjonalności jakie posiada portal wraz z podziałem na użytkowników zalogowanych i niezalogowanych dzięki którym spełnione zostały wszystkie wymagania funkcjonalne projektu. Warto tutaj zaznaczyć, że użytkownicy zalogowani, poza wypisanymi funkcjonalnościami posiadają też wszystkie funkcjonalności dostępne dla użytkowników niezalogowanych.

Funkcjonalności użytkowników niezalogowanych

- Przegląd postów i komentarzy
- Sortowanie wpisów po liczbie podbić
- Filtrowanie postów po typie postu oraz modelu i marce pojazdu
- Wyszukiwanie postów po tytule
- Możliwość logowania do portalu
- Możliwość rejestracji nowego konta w systemie
- Możliwość przypomnienia hasła za pomocą automatycznego przekierowania do adresu email administratora

Funkcjonalności użytkowników zalogowanych

- Edycja swoich danych
- Zmiana hasła
- Dodawanie nowych postów wraz ze zdjęciami
- Przegląd własnych postów
- Usuwanie i edytowanie swoich post
- Dodawanie komentarzy wraz ze zdjęciem
- Usuwanie własnych komentarzy
- Domyślne filtrowanie postów po marce i modelu przypisanego do konta pojazdu
- Przeglądanie swoich powiadomień i z ich poziomu przekierowanie do odpowiedniego postu
- Możliwość wylogowania się
- Możliwość korzystania z pomocy poprzez automatyczne przekierowanie do adresu email administratora

3. Technologie wykorzystane do implementacji portalu

Zaimplementowany system można podzielić na trzy główne moduły: baza danych, moduł backend oraz moduł frontend. Przy tworzeniu każdego z modułów wykorzystany został szereg technologii. Poniżej przedstawiono najważniejszą informację o każdym z modułów pod względem wykorzystanych technologii, frameworków i języków programowania.

Baza danych

Baza danych wykorzystywana w zaimplementowanym systemie jest relacyjną bazą danych i została zaimplementowana w języku SQL. Skorzystano tutaj z otwartego systemu zarządzania relacyjnymi bazami danych jakim jest PostgreSQL. Sama jego instancja jest przechowywana w kontenerze dzięki wykorzystaniu oprogramowania Docker.

Moduł backend

Moduł zawierający całą logikę systemu w tym zarządzanie użytkownikami, postami, powiadomieniami, komentarzami oraz pojazdami zaimplementowany został w języku programowania Java z narzędziami Apache Maven służącym do automatyzacji budowy oprogramowania. W tym module wykorzystano frameworki takie jak:

- Spring – zarządzanie zależnościami między obiektami, tworzenie kontrolerów REST'owych, dostęp do bazy danych, autoryzacja użytkowników itp.
- Commons IO – zarządzanie plikami w systemie
- JSON Web Token – obsługa tokenów,
- Javax.validation – walidacja danych
- Lombok – szereg adnotacji wspomagających implementację i utrzymanie systemu
- FlywayDB – implementacja bazy danych
- Javax.annotation – zestaw adnotacji przydatnych przy implementacji systemu

Moduł frontend

Moduł zawierający implementację warstwy prezentacji systemu stworzony został w języku TypeScript z wykorzystaniem trzech głównych frameworków: Node.js do uruchamiania aplikacji, Next.js do budowania aplikacji i ReactJS do implementacji portalu. Poniżej przedstawiono pozostałe frameworki wykorzystane w tym module:

- Axios – komunikacja restowa,
- Formik – obsługa formularzy,
- Primeflex, Prmelcons – gotowe ikony oraz style,
- Primereact – gotowe komponenty,
- React-dom – routowanie po systemie
- Universal-cookie – obsługa ciasteczek w przeglądarce
- Yup – walidacja danych.

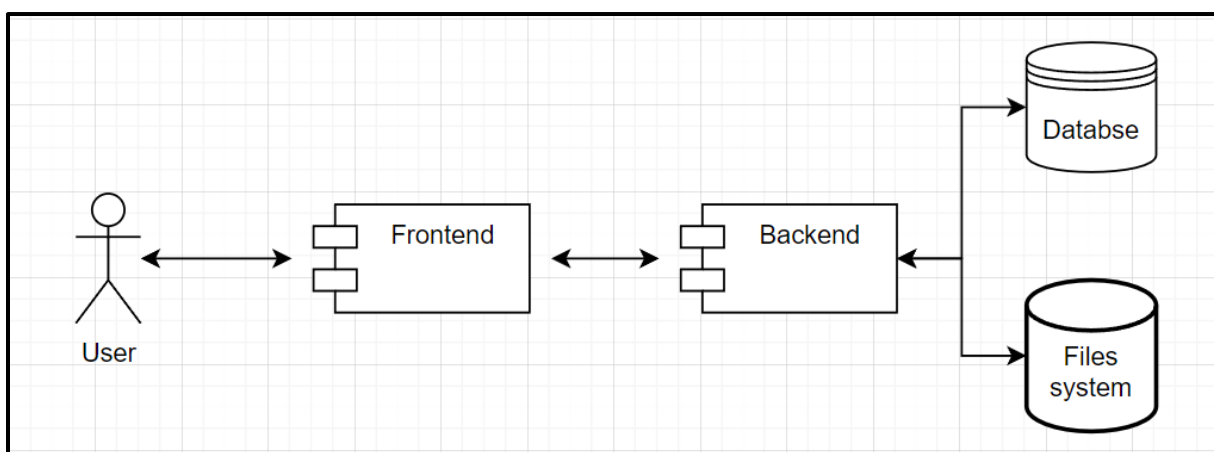
Pozostałe narzędzia

Poza wyżej wymienionym technologiami wykorzystano również język Git jako system kontroli wersji i platformę GitHub do hostowania repozytoriów. Ponadto do zarządzania pracą wykonywaną w ramach implementacji systemu wykorzystano narzędzie Trello w którym tworzone były zadania konieczne do wykonania aby spełnić wymagania postawione przed projektem. Sama implementacja odbywała się natomiast w IDE IntelliJ. Poniżej znajdują się link do repozytorium z kodem źródłowym:

[Repozytorium GitHub](#)

4. Implementacja systemu

Poniżej przedstawiono schemat zaimplementowanego systemu. Użytkownik korzysta z aplikacji, która jest zaimplementowana w module frontend. Moduł ten natomiast komunikuje się z warstwą zawierającą całą logikę systemu – modulem backend za pomocą protokołu REST. Posiada on połączenie z bazą danych na której może wykonywać wszystkie działań CRUD. Ponadto ma on dostęp do systemu plików na którym on jest zainstalowany i ma możliwość czytania, dodawania i usuwania plików z danych folderów.



Schemat nr 1 – schemat zaimplementowanego systemu

4.1. Implementacja systemu bazodanowego wykorzystane w systemie

W celu automatyzacji systemu bazodanowego skorzystano z systemu konteneryzacji Docker. Dzięki niemu w dość łatwy sposób jesteśmy w stanie uruchamiać system w kontenerze oraz nim zarządzać. Poniżej przedstawiono implementacją pliku *docker-compose.yml* wykorzystanego do tego celu, a także plik *init.sql*, który odpowiada za stworzenie bazy danych oraz użytkownika do jej zarządzania.

```

version: '3.9'
services:
  postgres:
    container_name: automotive-portal-db
    image: postgres:14-alpine
    ports:
      - 5432:5432
    environment:
      - POSTGRES_PASSWORD=ac
    volumes:
      - ~/apps/postgres:/var/lib/postgresql/data
      - ./init.sql:/docker-entrypoint-initdb.d/init.sql

```

Kod źródłowy nr 1 – implementacja pliku docker-compose.yml

```

create user ac with password 'ac';
alter user ac with SUPERUSER;
create database ac owner ac;
\connect ac;
create extension if not exists "uuid-ossf";

```

Kod źródłowy nr 2 – implementacja pliku init.sql

Jeśli chodzi o samą implementację struktury baz danych to została ona stworzona w module backend z wykorzystaniem frameworka flyway. Na kodzie źródłowym nr 3 przedstawiono jedną z implementacji pliku flyway, który zawiera pierwotną implementację tabeli z użytkownikami.

```

create schema if not exists ap;
CREATE TABLE ap.users
(
  id          uuid PRIMARY KEY,
  created_at  DATE    NOT NULL,
  modified_at DATE    NOT NULL,
  last_activity_at DATE NOT NULL,
  deleted     boolean NOT NULL,
  role        VARCHAR(100) NOT NULL,
  password    VARCHAR(200) NOT NULL,
  name        VARCHAR(100) NOT NULL,
  surname     VARCHAR(100) NOT NULL,
  email       VARCHAR(100) NOT NULL,
  phone_number VARCHAR(12),
  vehicle_id  bigint
);

```

Kod źródłowy nr 3 – przykładowy plik flyway'owy zawierający implementację tabeli users

Na poniższym diagramie ERD przedstawiono schemat opracowanej bazy danych, gdzie każda z tabel reprezentuje:

1. *Users* – użytkowników wraz z możliwością obsługi ról w przyszłości
2. *Posts* – wpisy użytkowników w systemie
3. *Comments* – komentarze do wpisów przez użytkowników
4. *Images* – zdjęcia dołączone do wpisów
5. *Notification* – powiadomienia użytkowników do danego postu
6. *Appearance* – tabela asocjacyjna łącząca użytkowników z postami (podbicia postów).

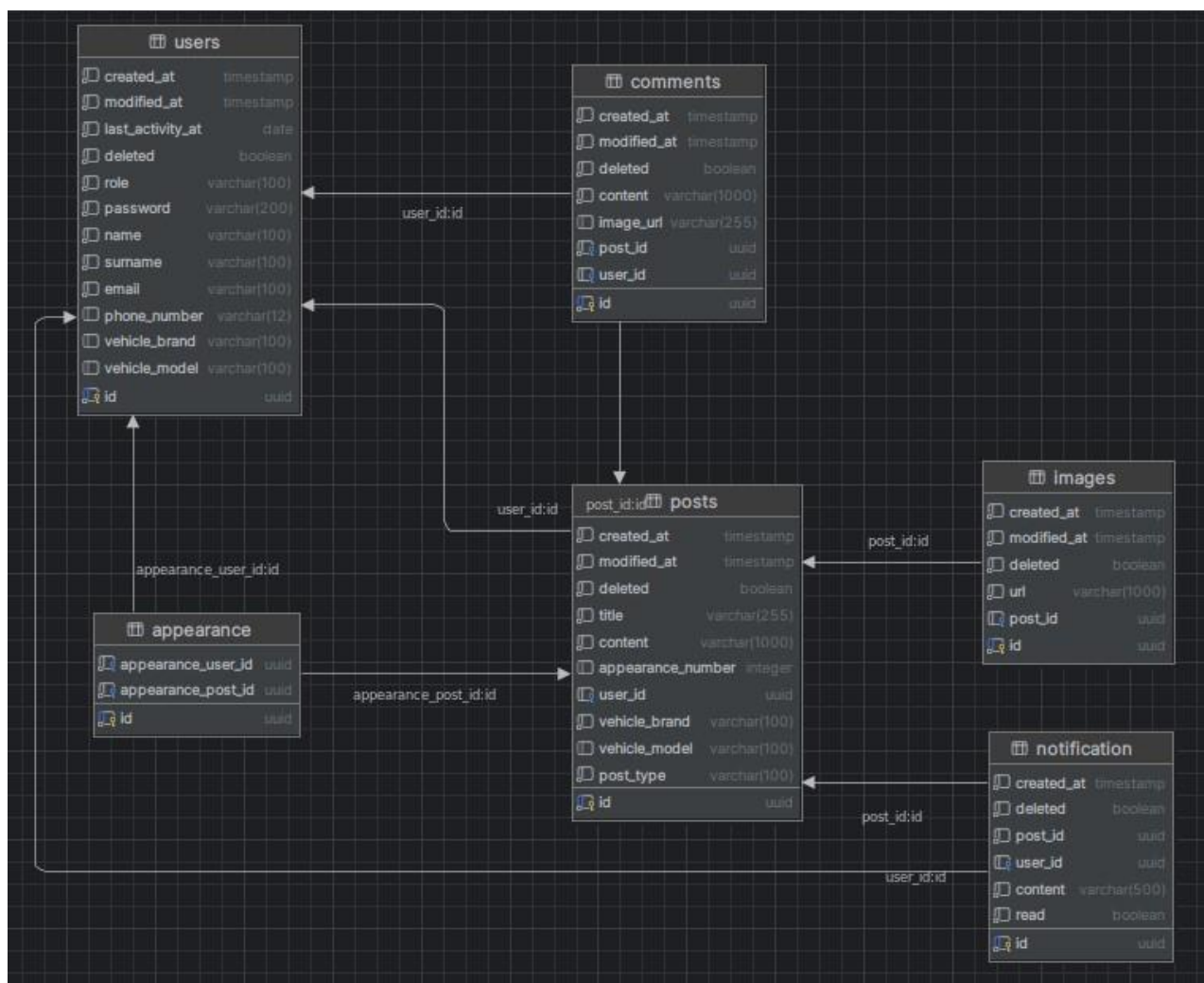


Diagram nr 1 – diagram ERD przedstawiający schemat zaimplementowanej bazy danych

4.2. Implementacja backendu

Jak już wspomniano wcześniej moduł backend odpowiada za całą logikę systemu w tym zapis i odczyt danych z bazy danych. Jego implementacja pozwala na obsługę wszystkich żądań przychodzących z frontendu. Każdy z requestów jest sprawdzany i w odpowiedni sposób autoryzowany za pomocą frameworka Spring Security. Poniżej znajduje się jedna z najważniejszych części kodu odpowiadająca za konfigurację przyjmowanych requestów, a dokładniej ich autoryzację.

```
@SuppressWarnings("deprecation")
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http.cors().and().csrf().disable()
        .httpBasic()
        .authenticationEntryPoint(new EntryPoint())
        .and()
        .authorizeHttpRequests()
        .requestMatchers("/auth/**", "/api/users/register", "api/vehicle/**").permitAll()
        .requestMatchers(HttpMethod.GET, "/api/comments/**").permitAll()
        .requestMatchers(HttpMethod.GET, "/api/posts/**").permitAll()
        .requestMatchers(HttpMethod.POST, "/api/posts/pageable").permitAll()
        .requestMatchers("/api/**").hasRole(UserRole.USER_ROLE.getRole())
        .anyRequest().denyAll()
        .and()
        .addFilterBefore(jwtAuthorizationFilter, UsernamePasswordAuthenticationFilter.class)
        .logout()
        .logoutSuccessHandler(new
            HttpStatusReturningLogoutSuccessHandler(HttpStatus.NO_CONTENT))
        .logoutUrl("/logout")
        .invalidateHttpSession(true)
        .deleteCookies("AUTOMOTIVEPORTALSESSIONID")
        .permitAll();
    return http.build();
}
```

Kod źródłowy nr 4 – konfiguracja requestów wymagających autoryzacji oraz ogólnodostępnych

Ponadto moduł obsługuje tworzenie kont w systemie, zmianę danych i hasła użytkowników oraz samo logowanie, które opiera się na generowaniu tokenów JWT. W celu uniknięcia problemu z częstą koniecznością logowania zaimplementowano również endpoint do odświeżania tokena. Poniżej znajduje się implementacja przykładowego endpointu - w tym przypadku przeznaczonego do logowania.

```

@ResponseBody
@RequestMapping(value = "/login", method = RequestMethod.POST)
public ResponseEntity login(@RequestBody AuthorizationRequestHeader
    authorizationRequestHeader) {
    try {
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                authorizationRequestHeader.getEmail(),
                authorizationRequestHeader.getPassword()));
        String email = authentication.getName();
        User user = new User(email, "");
        String token = jwtUtil.createToken(user);
        userService.updateLastActivityUser(email);
        UUID userId = userService.getUserIdByEmail(email);
        AuthorizationResponseHeader authorizationResponseHeader = new
            AuthorizationResponseHeader(email, token, userId);
        log.info("User {} has been logged.", email);
        return ResponseEntity.ok(authorizationResponseHeader);
    } catch (BadCredentialsException e) {
        ErrorResponse errorResponse = new ErrorResponse(HttpStatus.UNAUTHORIZED, "Invalid
            username or password");
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED).body(errorResponse);
    } catch (Exception e) {
        ErrorResponse errorResponse = new ErrorResponse(HttpStatus.BAD_REQUEST,
            e.getMessage());
        return ResponseEntity.status(HttpStatus.BAD_REQUEST).body(errorResponse);
    }
}

```

Kod źródłowy nr 5 – endpoint obsługujący logowanie w systemie

Backend odpowiada również za obsługę wszystkich operacji jakie może wykonać użytkownik do których można zaliczyć operacje takie jak: dodawanie / edycja / usuwanie postów, dodawanie / usuwanie komentarzy czy wczytywanie postów wraz z możliwością sortowania po użytkowniku i liczbie podbić postu oraz filtracją po jego typie, a także marce i modelu pojazdów. Każdy z parametrów przekazanych do endpointów jest walidowany za pomocą odpowiednich adnotacji w obiektach DTO's. Ponadto moduł obsługuje również funkcjonalność wyszukiwania postów po tytule. Warto tutaj zaznaczyć, że nie zwraca on wszystkich znalezionych postów, lecz tylko taką ilość jaka jest przekazana jako parametr. Na fragmencie kodu źródłowego nr 6 przedstawiono przykładową implementację tych funkcjonalności – w tym przypadku endpoint obsługujący dodawanie postu.

```

@PostMapping(consumes = MediaType.MULTIPART_FORM_DATA_VALUE)
public void createPost(@RequestPart @Valid PostFormDTO postFormDTO,
@RequestPart(value = "images", required = false) MultipartFile[] images) {
    Authentication loggedInUser = SecurityContextHolder.getContext().getAuthentication();
    String username = loggedInUser.getName();
    UUID postId = UUID.randomUUID();
    List<Image> images1 = new ArrayList<>();
    if (images != null) {
        for (MultipartFile image : images) {
            images1.add(imageService.createImage(image, postId));
        }
    }
    postService.createPost(postFormDTO, username, images1, postId);
    log.info("Created new post by {} with {} images", username, images == null ? "0" :
        images.length);
}

```

Kod źródłowy nr 6 – implementacja endpointu do obsługi dodawania postu

Moduł obsługuje także możliwość zapisu i usuwania zdjęć przypisanych do postów i komentarzy w systemie plików (fragment kodu źródłowego nr 7 – w tym przypadku usuwanie zdjęć).

```

@Transactional
public void removeImages(List<String> imagesToRemove, Post post) throws IOException {
    List<File> filesToRemove = imagesToRemove.stream().map((img) -> new
        File(String.valueOf(createPathToImage(post.getId().toString(), img))))).toList();
    List<Image> images = imageRepository.findAllByPost(post);
    for (File file : filesToRemove) {
        file.delete();
    }
    if (images.size() == filesToRemove.size()) {
        FileUtils.deleteDirectory(new File(uploadDirectory + post.getId().toString()));
    }
    log.info("Deleted {} files", filesToRemove.size());
}

```

Kod źródłowy nr 7 – funkcja do usuwania zdjęć z systemu plików

Moduł posiada również implementację obsługi powiadomień, które są tworzone dla użytkowników w przypadku gdy ktoś skomentuje jego post oraz ich odpowiednie oznaczanie w przypadku odczytania powiadomienia przez użytkownika. Ponadto w celu usuwania starych powiadomień (np. sprzed 5 dni – wartość konfigurowalna w ustawieniach) w module znajduje się cron, który uruchamia się każdego dnia o wyznaczonej godzinie (fragment kodu źródłowego nr 8).

```

@Scheduled(cron = "${ap.deleted.old.notification.cron}")
private void deleteOldNotifications() {
    int deleteNotificationsNumber = notificationService.deleteOldNotifications();
    log.info("Delete {} old notifications", deleteNotificationsNumber);
}

```

Kod źródłowy nr 8 – cron do usuwania starych powiadomień

Modele oraz marki pojazdów pobierane są natomiast z zewnętrznego [API](#) czego implementacja również znajduje się w tym module. Poniżej znajduje się jedna z funkcji do obsługi tej funkcjonalności – w tym przypadku pobieranie wszystkich marek pojazdów.

```

public List<VehicleBrandDTO> getAllBrands() {
    boolean downloadAllRows = true;
    int offsetRows = 0;
    List<VehicleBrandDTO> allBrands = new ArrayList<>();
    while (downloadAllRows) {
        String allBrandsUrl = ALL_BRANDS_URL.replace("{limit}",
            String.valueOf(requestLimit)).replace("{offset}",
String.valueOf(offsetRows));
        HttpRequest request =
HttpRequest.newBuilder().uri(URL.create(allBrandsUrl)).build()
        try {
            HttpResponse<String> response = client.send(request,
                HttpResponse.BodyHandlers.ofString());
            if (response.statusCode() == 200) {
                String responseBody = response.body();
                VehicleBrandResultsDTO vehicleBrandResultsDTO =
                    mapper.readValue(responseBody, VehicleBrandResultsDTO.class);
                allBrands.addAll(vehicleBrandResultsDTO.results());
                if (vehicleBrandResultsDTO.results().size() == requestLimit) {
                    offsetRows += requestLimit;
                } else {
                    downloadAllRows = false;
                }
            } else {
                throw new BadRequestException("Wrong url to external service");
            }
        } catch (IOException | InterruptedException exception) {
            throw new BadRequestException("Wrong url to external service");
        }
    }
    return allBrands;
}

```

Kod źródłowy nr 9 – pobieranie marek pojazdów z zewnętrznego API

4.3. Implementacja frontendu

Moduł frontend odpowiada za warstwę prezentacji systemu oraz obsługę wszystkich zdarzeń wykonywanych przez użytkownika. Jego implementacja gwarantuje spełnienie wszystkich funkcjonalności przedstawionych w rozdziale nr 2. Komunikuje on się z modułem backend za pomocą protokołu REST. Cała aplikacja jest responsywna i skaluje się w odpowiedni sposób na urządzeniach mobilnych co zostało przedstawione w rozdziale 5 z prezentacją systemu.

Jedną z ważniejszych funkcjonalności jakie posiada jest możliwość na bieżąco pobierania kolejnych postów w przypadku scrollowania strony przez użytkownika i dojściu do ostatniego wczytanego wpisu. Do tego celu wykorzystany został Intersection observer, która odpowiada za sprawdzanie czy na ekranie pojawił się ostatni wczytany post, jeśli tak to wywołuje on funkcję odpowiadającą za wczytanie kolejnych postów z backendu – po ustawieniu nowej strony wczytywane są nowe dane (fragment kodu źródłowego nr 10).

```
useEffect(() => {
  if (observer.current) observer.current.disconnect();
  const options = {
    root: null,
    rootMargin: '0px',
    threshold: 0.5
  };
  observer.current = new IntersectionObserver((entries) => {
    if (entries[0].isIntersecting && !loading) {
      setPage((prevPage) => prevPage + 1);
    }
  }, options);
  if (loadMoreRef.current) {
    observer.current.observe(loadMoreRef.current);
  }
  return () => {
    if (observer.current) observer.current.disconnect();
  };
}, [loadMoreRef.current]);
```

Kod źródłowy nr 10 – implementacja observera do sprawdzania czy na stronie pojawił się ostatni wczytany post

Samo wywołanie endpointu wczytującego kolejne posty zostało przedstawione natomiast na poniższym fragmencie kodu. Instrukcje te uwzględniają również filtry oraz sortowanie.

```
const loadPosts = async (requirePage: number) => {
  const newPosts = await getPageablePosts({
    page: requirePage,
    size: 5,
    searchValue: searchValue === "" ? null : searchValue,
    sortByAppearanceNumber: sortPostsByAppearanceNumber,
    userId: showMyPosts ? (user ? user.id.slice(0, user.id.length) : null) : null,
    vehicleBrand: selectedVehicleBrand,
    vehicleModel: selectedVehicleModel,
    postType: selectedPostType
  });
  return newPosts.data;
};
```

Kod źródłowy nr 11 – wczytywanie postów

Moduł obsługuje również automatyczne odświeżanie powiadomień co minutę bez konieczności odświeżania strony. Każdy z formularzy (dodawanie i edycja post, dodawanie komentarza, tworzenie konta, zmiana hasła i danych użytkownika) podlega walidacji również w tym module – przed wysłaniem żądania na backend. Na poniższym kodzie źródłowym przedstawiono przykładową walidację – w tym dla formularza z dodawaniem postu.

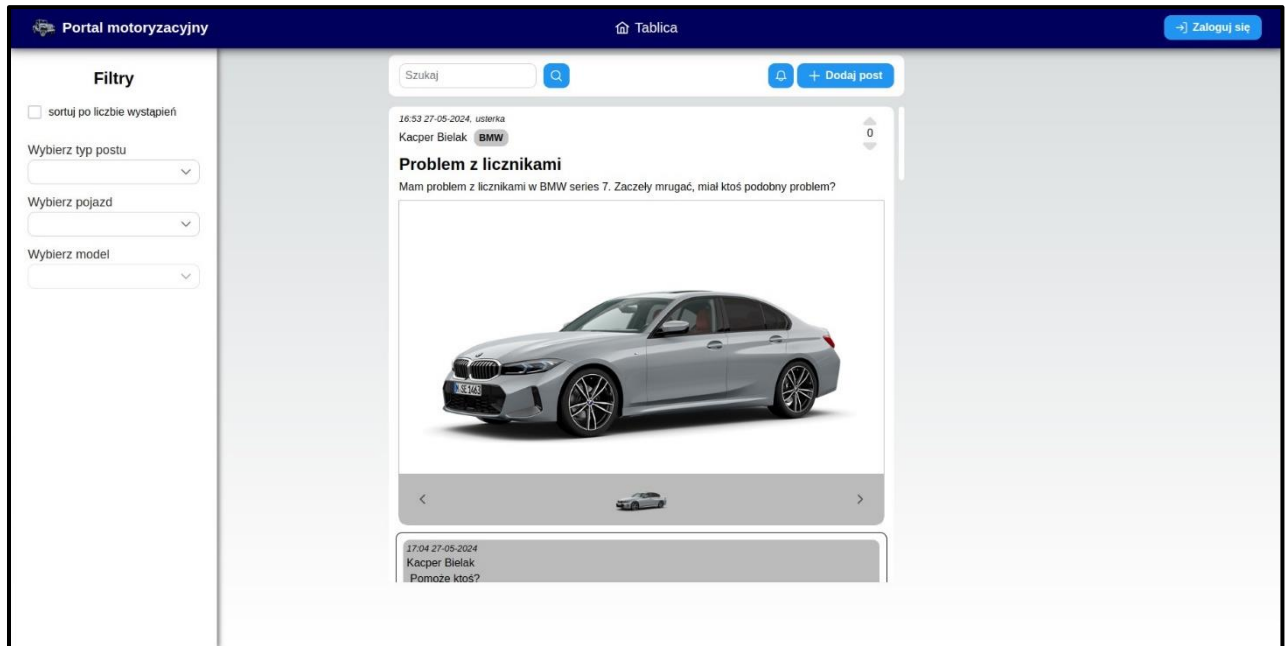
```
export const postDataValidation: Yup.SchemaOf<PostFormDTO> = Yup.object().shape({
  title: Yup.string().required("Pole wymagane").max(100, "Tytuł jest za długi"),
  content: Yup.string().nullable().required("Pole wymagane"),
  vehicleBrand: Yup.string().nullable().required("Pole wymagane"),
  postType: Yup.string().nullable().required("Pole wymagane"),
  vehicleModel: Yup.string().nullable().notRequired()
});
```

Kod źródłowy nr 12 – walidacja formularza dodawania postu

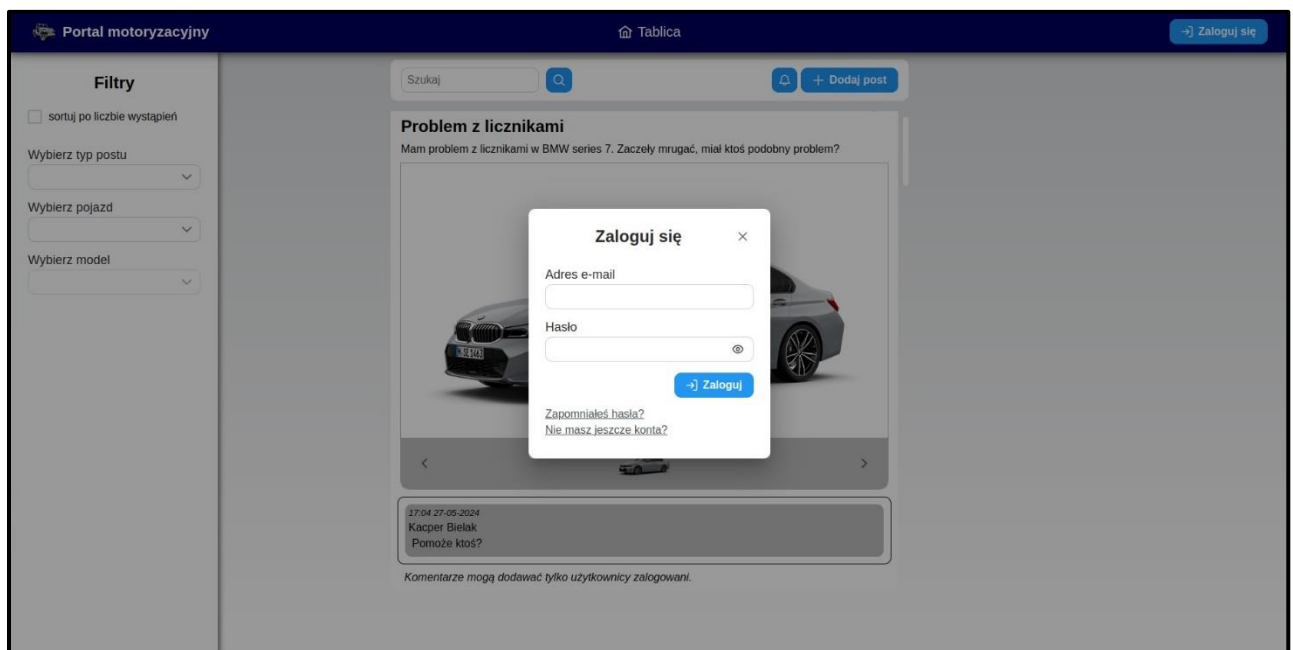
5. Prezentacja stworzonego systemu

Poniżej przedstawiono zrzuty ekranu z zaimplementowanego systemu. Każdy ze screenów został opisany. Dodatkowo na ostatnim zrzucie ekranu przedstawiono przykładowy widok na urządzeniu mobilnym.

Główny widok dla użytkownika niezalogowanego



Panel logowania



Panel rejestracji

Portal motoryzacyjny

Załącz konto

Adres e-mail*

Imię*

Nazwisko*

Numer telefonu

Wybierz pojazd

Wybierz model

Hasło*

Potwierdź hasło*

* - pole wymagane

Wyjdź

Zarejestruj

Widok po zalogowaniu z sortowaniem postów po liczbie podbić i aktywnym powiadomieniem

Portal motoryzacyjny

Tablica

kacperbielak123@o2.pl

Filtry

sortuj po liczbie wystąpień

moje posty

Wybierz typ postu

Wybierz pojazd

Wybierz model

Szukaj

+ Dodaj post

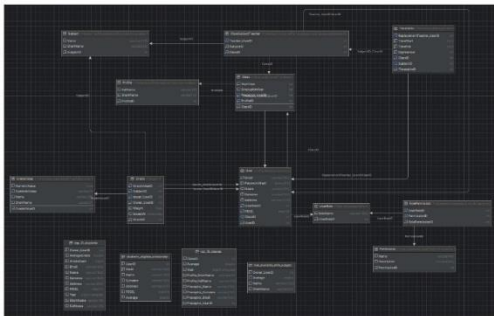
12:39 25-05-2024, Kupię

Jan Kowalski BMW 328i Coupe

1

Kupię BMW E90

Kupię bmw bla bla



Powiadomienia

16:25 28-05-2024 NOWE

Użytkownik Jan Kowalski dodał komentarz do twójgo postu.

przejdź do postu

16

Widok po zalogowaniu z przykładowym filtrowaniem po pojeździe

The screenshot shows the 'Portal motoryzacyjny' interface. On the left, the 'Filtry' section includes checkboxes for 'sortuj po liczbie wystąpień' and 'moje posty', and dropdown menus for 'Wybierz typ postu', 'Wybierz pojazd' (set to 'Audi'), and 'Wybierz model' (set to 'S5'). The main content area features a search bar with 'Szukaj' and a '+ Dodaj post' button. Below the search bar, two example posts are displayed. The first post, dated '20:15 27-05-2024', is by 'Kacper Bielak' and titled 'Przykładowy tytuł usterki'. It includes a description 'Usterka samochodu audi s5, opis ponizej.' and a comment section with the text 'brak komentarzy' and a 'Dodaj komentarz' input field. The second post, dated '20:26 25-05-2024', is also by 'Kacper Bielak' and titled 'Przykładowy tytuł usterki'. It includes a description 'asd' and a table titled 'Class 9c'.

Przebieg	Class 9c	Stosunek
10000 km	10000 km	10000 km
20000 km	20000 km	20000 km
30000 km	30000 km	30000 km
40000 km	40000 km	40000 km
50000 km	50000 km	50000 km
60000 km	60000 km	60000 km
70000 km	70000 km	70000 km
80000 km	80000 km	80000 km
90000 km	90000 km	90000 km
100000 km	100000 km	100000 km

On the right, the 'Powiadomienia' section shows a notification from 'Użytkownik Jan Kowalski' dated '16:25 28-05-2024', stating 'dodał komentarz do twojego postu.' with a 'przejdź do postu' link.

Obsługa szukania po tytule

The screenshot shows the 'Portal motoryzacyjny' interface with the search bar containing 'Kupie'. The main content area displays a search result for 'Kupie BMW E90' by 'Jan Kowalski', dated '12:39 25-05-2024'. The post includes a description 'Kupie bmw bla bla' and a large image of a BMW E90. The right sidebar shows a notification from 'Użytkownik Jan Kowalski' dated '16:25 28-05-2024', stating 'dodał komentarz do twojego postu.' with a 'przejdź do postu' link.

Widok dodawania postu

Portal motoryzacyjny

Tablica

kacperbielak123@o2.pl

Filtry

☒ sortuj po liczbie wystąpień
☐ moje posty

Wybierz typ postu

Wybierz pojazd

Wybierz model

Nowy post

Typ postu*

Tytuł*

Opis*

Wybierz pojazd*

Audi

Wybierz model

S5

0 / 2

Brak załączników

* - pole wymagane

Anuluj Dodaj

Powiadomienia

16:25 28-05-2024 **NOWE**
Użytkownik Jan Kowalski dodał komentarz do twojego postu.
[przejdź do postu](#)

Widok edycji postu

Portal motoryzacyjny

Tablica

kacperbielak123@o2.pl

Filtry

☒ sortuj po liczbie wystąpień
☐ moje posty

Wybierz typ postu

Wybierz pojazd

Wybierz model

Nowy post

Typ postu*

ogólny

Tytuł*

Pytanie odnośnie alfa romeo

Opis*

Chce zakupić alfa romeo. Czy znacie najczęstsze usterki?

Wybierz pojazd*

Alfa Romeo

Wybierz model

1 / 2

Alfa

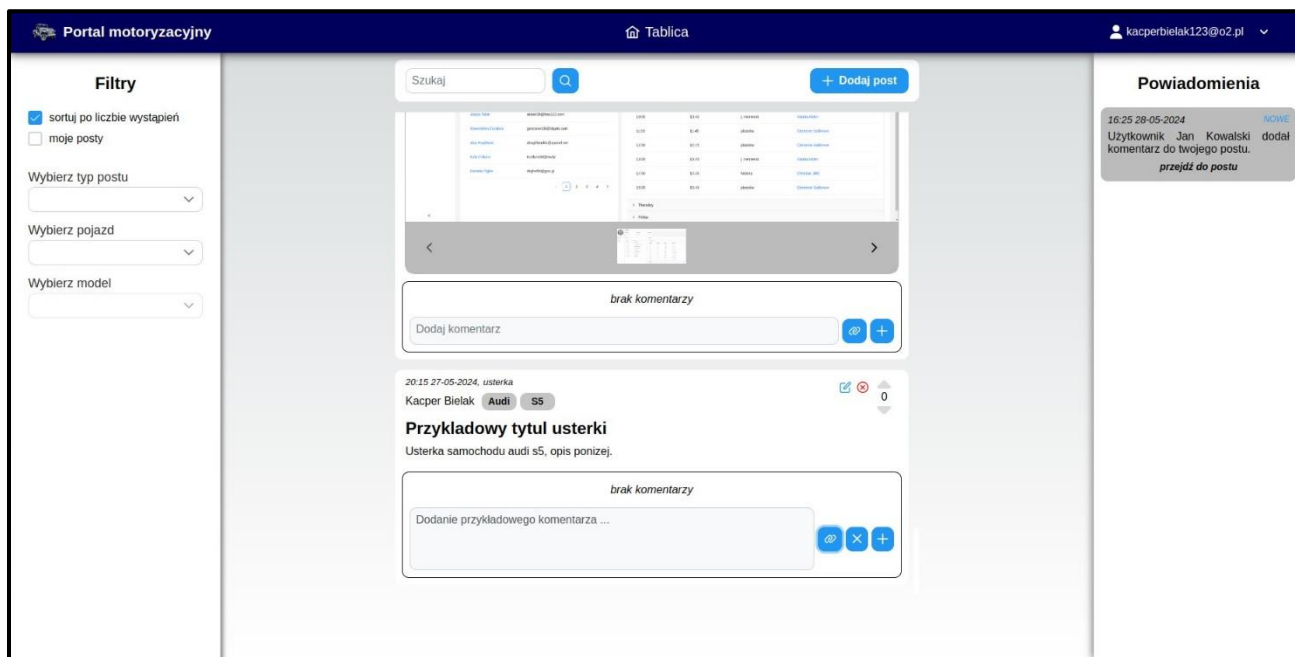
* - pole wymagane

Anuluj Dodaj

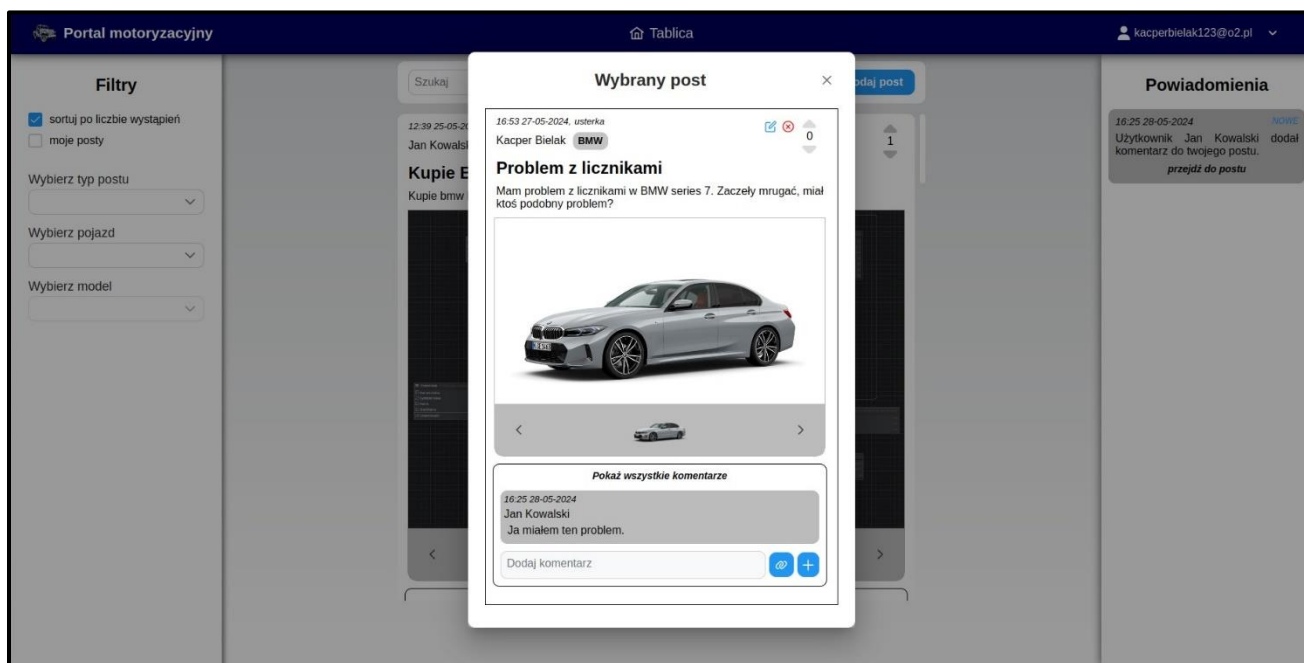
Powiadomienia

16:25 28-05-2024 **NOWE**
Użytkownik Jan Kowalski dodał komentarz do twojego postu.
[przejdź do postu](#)

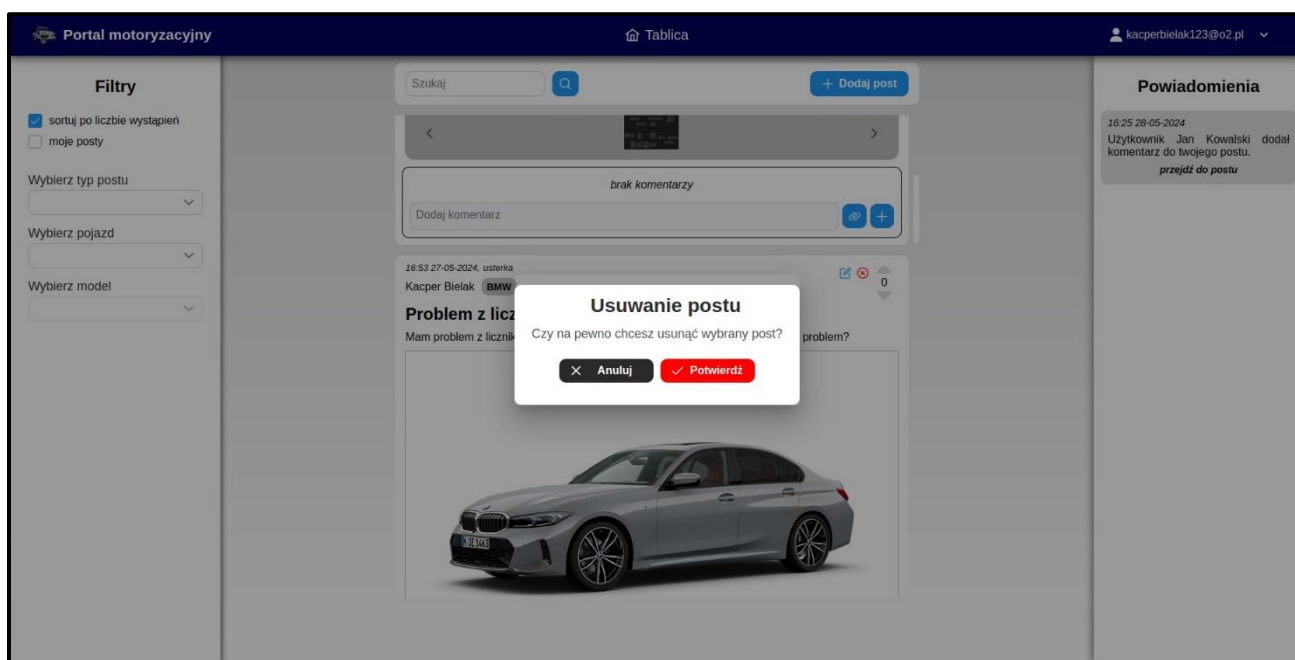
Dodawanie komentarza do postu



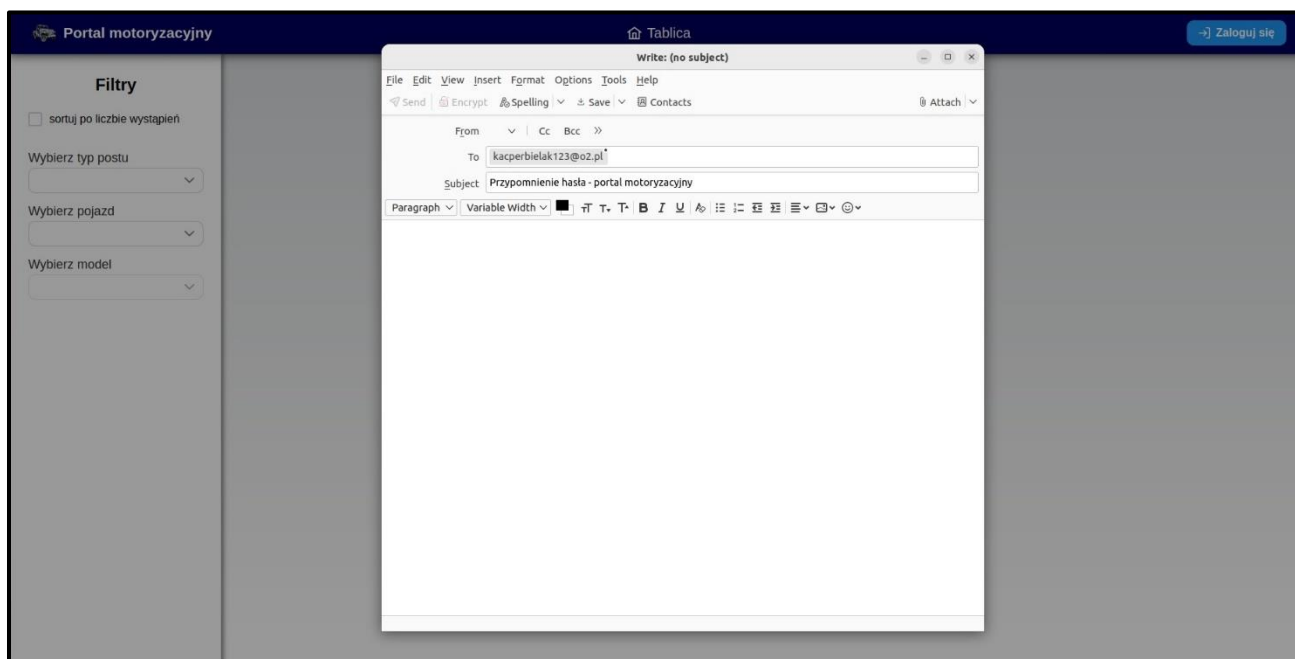
Widok postu z poziomu powiadomienia



Potwierdzenie usunięcia – w tym przypadku postu



Obsługa przypomnienie hasła lub pomocy – przekierowanie do tworzenia maila



Widok zmiany danych użytkownika lub hasła

Portal motoryzacyjny

Tablica

kacperbielak123@o2.pl

Twoje dane

Adres e-mail*

kacperbielak123@o2.pl

Imię*

Kacper

Nazwisko*

Bielak

Numer telefonu

603774322

Wybierz pojazd

Audi

Wybierz model

S5

* - pole wymagane

Zmień hasło

Edytuj dane

Główny widok na urządzeniach mobilnych

[illegible]

6. Wnioski i możliwości dalszego rozwoju

Zaimplementowany system będący portalem motoryzacyjnym jest świetną opcją dla osób szukających odpowiedzi na różne pytania odnoszące się do danego pojazdu lub samej marki. Dzięki możliwości przeglądania i odpowiedniego filtrowania już istniejących postów jest w stanie w łatwy i szybki sposób znaleźć odpowiedź na jego pytania. Ponadto, jeśli mu się to nie uda to może sam dodać post i czekać na odpowiedź innych osób o czym będą go informować powiadomienia. Dodatkowo taki system może wspomóc użytkowników przy zakupie pojazdu. Dzięki filtrowaniu po typie postu oraz liczbie podbić może szybko znaleźć najczęstsze usterki danego modelu co ułatwi weryfikację stanu konkretnego pojazdu przy jego zakupie. Opracowany portal motoryzacyjny jest zdecydowanie lepszą opcją niż istniejące dotychczas grupy na facebooku gdzie ciężko jest znaleźć konkretne informacje na temat konkretnych pojazdów.

Stworzony portal motoryzacyjny jest gotową wersją do testowania przez użytkowników. Może zostać wdrożony na serwer testowy i udostępniony pewniej grupie osób, która pełniła by rolę testerów i zgłaszała swoje uwagi. Do finalnej wersji systemu konieczne wydaje się być wprowadzenie serwera pocztowego do weryfikacji kont. Ponadto przydatne byłoby opracowanie panelu administracyjnego gdzie admin mógłby zarządzać użytkownikami oraz ich treściami – system jest przygotowany do obsługi ról. Jeśli chodzi o same funkcjonalności z w przyszłości można by było dodać widok profilu użytkowników oraz opracować funkcjonalność wymiany prywatnych wiadomości między nimi. Dodatkowo warto byłoby dodać zapisywanie postów do ulubionych oraz zgłaszanie nieodpowiednich postów, które przekazywanej by były do weryfikacji przez administratora.