

Wiktor Majchrzak , Maciej Kuczyński

# Dokumentacja Mars-Dust

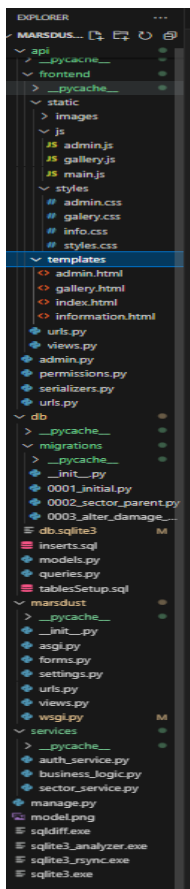
## Wstęp

Mars-Dust to projekt przedstawiający model 3D Marsa oraz dane dotyczące burz piaskowych na tej planecie. Dokumentacja zawiera informacje niezbędne do uruchomienia aplikacji i korzystania z jej funkcji. Zawarto opis działania kodu oraz sposób obsługi aplikacji. Projekt ma na celu wizualizację i analizę burz piaskowych na Marsie oraz ich wpływu na misje kosmiczne.

Link do projektu <https://github.com/kuczynskimaciej1/MarsDust>

## Struktura

Projekt Mars-Dust składa się z trzech głównych sekcji: frontendu, bazy danych i głównego katalogu aplikacji. Frontend zawiera statyczne zasoby, takie jak obrazy (np. tekstury Marsa i jego księżyców), skrypty JavaScript oraz szablony HTML do renderowania interfejsu użytkownika. Baza danych obejmuje modele Django, migracje oraz niestandardowe zapytania SQL do zarządzania danymi. Główny katalog marsdust zawiera konfigurację projektu, logikę biznesową w katalogu services oraz pliki zarządzające, takie jak manage.py. Aplikacja jest modularna, co ułatwia rozwój i utrzymanie, a jej struktura obejmuje również narzędzia do zarządzania bazą danych SQLite.



## Plik Index.html

Plik index.html jest głównym plikiem frontendowym aplikacji, odpowiedzialnym za wyświetlanie interfejsu użytkownika. Zawiera strukturę HTML, która ładuje wszystkie istotne zasoby, takie jak style CSS i obrazy, dzięki użyciu tagu `{% load static %}`. W pliku zaimplementowano także zmienną `csrfToken`, zapewniającą bezpieczeństwo przy wysyłaniu formularzy. Na stronie znajduje się nawigacja z przyciskami umożliwiającymi logowanie, rejestrację oraz przejście do galerii i sekcji informacyjnej. Istnieją również przyciski do wyświetlenia danych o Marsie oraz księżycach Phobosie i Deimosie. Panel logowania i rejestracji pozwala użytkownikowi na zalogowanie się lub założenie konta. Po zalogowaniu dostępne są panele z informacjami o sektorze Marsa, a także przyciski do wyświetlania statusu bazy. Dla niezalogowanych użytkowników dostępny jest prosty panel z informacjami o sektorach. Istnieją także dedykowane panele dla Phobosa i Deimosa, które wyświetlają szczegóły na temat tych dwóch księżyców Marsa. W pliku znajduje się również sekcja ładowania tekstur, dzięki której wyświetlane są obrazy Marsa, Phobosa i Deimosa. Skrypt `three.js` jest załadowany z CDN, co umożliwia renderowanie grafiki 3D, a główny skrypt `main.js` odpowiada za zarządzanie interakcjami użytkownika, w tym nawigacją po modelu i obsługą paneli informacyjnych. Plik index.html jest więc kluczowym elementem aplikacji, łączącym wszystkie jej funkcje i zapewniającym interakcję z użytkownikiem.

```
api > frontend > templates > index.html > html > body
1  {% load static %}
2  <DOCTYPE html>
3  <!-- Inside index.html -->
4  <script>
5  |   const csrfToken = "{{ csrf_token }}";
6  </script>
7
8  <html lang="pl">
9  <head>
10 |   <meta charset="UTF-8" />
11 |   <meta name="viewport" content="width=device-width, initial-scale=1.0" />
12 |   <title>Mars - 3D Model</title>
13 |   <link rel="stylesheet" href="{% static 'styles/styles.css' %}" />
14 </head>
15 <body>
16 |   <nav id="navbar">
17 |     <div id="app-name">Mars-Dust</div>
18 |     <div id="auth-buttons">
19 |       <button id="login-button">Zaloguj się</button>
20 |       <button id="register-button">Zarejestruj</button>
21 |       <a href="{% url 'gallery' %}"><button id="gallery-button">Galeria</button></a>
22 |       <a href="{% url 'information' %}"><button id="info-button">Informacja</button></a>
23 |       <!-- Przycisk do Phobos -->
24 |       <button id="phobos-button">Phobos</button>
25 |       <!-- Przycisk do Deimos -->
26 |       <button id="deimos-button">Deimos</button>
27 |     </div>
28 |   </nav>
29
30 |   <div id="mars-container"></div>
31
32 |   <div id="controls">
33 |     <button id="prev-sector"></button>
34 |     <div id="sector-buttons-container">
35 |       <div id="sector-buttons">
36 |         <!-- Przycisk sektora będzie dynamicznie generowany przez JavaScript -->
37 |       </div>
38 |     </div>
39 |     <button id="next-sector"></button>
40 |   </div>
41
42
```

```

<!-- Panel z informacjami o sektorze dla niezalogowanych -->
<div id="info-panel-logout">
  <div id="info-content-logout">
    <h2>Informacje o sektorze</h2>
    <p id="info-text-logout">Wybierz sektor, aby zobaczyć szczegóły.</p>
    <!-- Przycisk Baza, który wyświetli dodatkowe informacje -->
    <button id="base-button">Status bazy</button>
  </div>
  <button id="close-panel-logout">Zamknij</button>
</div>

<!-- Panel z informacjami o sektorze dla zalogowanych -->
<div id="info-panel-login">
  <div id="info-content-login">
    <h2>Informacje</h2>
    <p id="info-text-login">Wybierz sektor, aby zobaczyć szczegóły.</p>
  </div>
  <button id="close-panel-login">Zamknij</button>
</div>

<div id="login-panel">
  <h2>Logowanie</h2>
  <form>
    <input type="text" placeholder="Nazwa użytkownika" required />
    <input type="password" placeholder="Hasło" required />
    <button type="submit">Zaloguj się</button>
  </form>
  <div id="login-actions">
    <button id="close-login-panel">Zamknij</button>
    <button id="admin-panel-button">Panel administratora</button>
  </div>
</div>

```

```

<div id="register-panel">
  <h2>Rejestracja</h2>
  <form>
    <input type="text" placeholder="Nazwa użytkownika" required />
    <input type="email" placeholder="Adres e-mail" required />
    <input type="password" placeholder="Hasło" required />
    <input type="password" placeholder="Powtórz hasło" required />
    <button type="submit">Zarejestruj się</button>
  </form>
  <button id="close-register-panel">Zamknij</button>
</div>

<!-- Panel z informacjami o Phobosie -->
<div id="phobos-panel">
  <div id="phobos-content">
    <h2>Phobos</h2>
    <p id="phobos-text">Wczytywanie danych o Phobosie...</p>
  </div>
  <button id="close-phobos-panel">Zamknij</button>
</div>

<!-- Panel z informacjami o Deimosie -->
<div id="deimos-panel">
  <div id="deimos-content">
    <h2>Deimos</h2>
    <p id="deimos-text">Wczytywanie danych o Deimosie...</p>
  </div>
  <button id="close-deimos-panel">Zamknij</button>
</div>

```

```

<!-- Wczytanie ścieżki do obrazu tekstury -->
<script>
  window.imagePath = "{% static 'images/mars-texture.jpg' %}";
</script>
<script>
  window.phobosTexture = "{% static 'images/phobos-texture.jpg' %}";
  window.deimosTexture = "{% static 'images/deimos-texture.jpg' %}";
</script>

<script src="https://cdn.jsdelivr.net/npm/three.js@r128/three.min.js"></script>
<script src="{% static 'js/main.js' %}"></script>

</body>
</html>

```

## Plik information.html

Plik information.html jest częścią aplikacji internetowej projektu Mars-Dust, która prezentuje informacje o burzach piaskowych na Marsie oraz celach projektu. Strona zawiera nagłówki i akapity, które w sposób przystępny przedstawiają szczegóły dotyczące zjawiska burz piaskowych na Marsie. W nagłówku dokumentu załadowany jest zewnętrzny plik CSS, który odpowiada za stylizację strony. W sekcji nawigacyjnej znajdują się linki do głównych sekcji aplikacji, takich jak Strona Główna, Galeria i Informacje, umożliwiające łatwą nawigację po projekcie. W treści strony znajduje się wprowadzenie do projektu, opis burz piaskowych oraz ich wpływu na Marsa, a także cel edukacyjny aplikacji, który polega na wizualizacji zjawisk atmosferycznych. Zawarto tu również informacje o autorach projektu: Wiktorze Majchrzaku i Macieju Kuczyńskim. Struktura strony została zaprojektowana w sposób przejrzysty, z odpowiednimi nagłówkami, co umożliwia łatwe przyswajanie informacji. Dzięki użyciu Django, plik korzysta z systemu szablonów, ładowania statycznych zasobów, takich jak pliki CSS. Plik information.html pełni kluczową rolę informacyjną, zapewniając użytkownikom dostęp do podstawowych informacji o projekcie. Zawiera treści edukacyjne oraz pomaga w zapoznaniu się z celami i założeniami projektu.

```
{% load static %}
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Informacje o projekcie</title>
    <link rel="stylesheet" href="{% static 'styles/info.css' %}" />
  </head>
  <body>
    <nav id="navbar">
      <div id="app-name">Mars-Dust</div>
      <div id="auth-buttons">
        <a href="{% url 'home' %}">
          <button id="home-button">Strona Główna</button></a>
        <a href="{% url 'gallery' %}">
          <button id="gallery-button">Galeria</button></a>
        <a href="{% url 'information' %}">
          <button id="info-button">Informacja</button></a>
      </div>
    </nav>

    <div id="content">
      <h1>Informacje o projekcie Mars-Dust</h1>
      <p>
        Witamy na stronie projektu Mars-Dust, stworzonego w ramach zajęć.
        Projekt ten przedstawia model 3D planety Marsa oraz informacje wraz z
        danymi związane z burzami piaskowymi na tej planecie.
      </p>
    </div>
  </body>
</html>
```

```

<div id="content">
  <h1>Informacje o projekcie Mars-Dust</h1>
  <p>
    Witamy na stronie projektu Mars-Dust, stworzonego w ramach zajęć.
    Projekt ten przedstawia model 3D planety Marsa oraz informacje wraz z
    danymi związane z burzami piaskowymi na tej planecie.
  </p>

  <h2>Czym są burze piaskowe na Marsie?</h2>
  <p>
    Burze piaskowe na Marsie to zjawiska atmosferyczne polegające na
    unoszeniu cząsteczek pyłu i piasku w atmosferze planety. Mogą one
    obejmować różne obszary, od lokalnych po globalne burze, które mogą
    objąć całą planetę. Mars posiada cienką atmosferę, która sprzyja
    tworzeniu silnych wiatrów, które unoszą pył i drobny piasek. Zjawisko to
    jest szczególnie nasilone w okresie letnim na Marsie, gdy występują duże
    zmiany temperatury, co powoduje różnice ciśnienia. Burze piaskowe mogą
    trwać od kilku dni do nawet kilku tygodni. Podczas ich występowania
    widoczność na Marsie znacznie się zmniejsza, a temperatura na
    powierzchni może ulec zmianie. Globalne burze piaskowe potrafią zakłócić
    działanie robotów marsjańskich, takich jak Opportunity, które straciły
    łączność w wyniku takich zjawisk. Przemieszczający się pył może również
    wpływać na badania naukowe, zmieniając warunki atmosferyczne. Cząsteczki
    piasku na Marsie są bardzo drobne, co ułatwia ich unoszenie w
    atmosferze, nawet mimo jej rzadkości. Burze piaskowe stanowią jedno z
    charakterystycznych zjawisk marsjańskiego klimatu i są istotnym
    wyzwaniem dla przyszłych misji na tę planetę.
  </p>

  <h2>Cel projektu</h2>
  <p>
    Projekt "Mars-Dust" ma na celu edukację użytkowników o charakterystyce
    marsjańskich burz piaskowych, ich przyczynach oraz wpływie na badania
    naukowe i misje kosmiczne. Dzięki wizualizacji tych danych, użytkownicy
    będą mogli lepiej zrozumieć dynamikę atmosferyczną Marsa i wyzwania
    związane z jego eksploracją.
  </p>

  <h2>Autorzy</h2>
  <p>Wiktor Majchrzak i Maciej Kuczyński</p>
</div>
</body>
html>

```

## Plik galery.html

Plik galery.html jest odpowiedzialny za wyświetlanie galerii zdjęć pobranych z API NASA w ramach aplikacji Mars-Dust. Strona ładuje odpowiednie zasoby CSS, aby zapewnić odpowiednią stylistykę galerii, przy użyciu tagu `{% load static %}`. W sekcji nawigacyjnej znajduje się przycisk prowadzący do Strony Głównej, a główną funkcjonalnością strony jest wyświetlanie galerii zdjęć związanych z Marsiem i przestrzenią kosmiczną. Sekcja main zawiera tytuł oraz kontener, w którym będą wyświetlane zdjęcia. Przed załadowaniem zdjęć użytkownik widzi komunikat Ładowanie zdjęć. W pliku znajduje się także sekcja lightbox, która umożliwia powiększenie zdjęcia po jego kliknięciu. Zawiera ona ukrytą na początku wersję zdjęcia, które wyświetlane jest w pełnym rozmiarze, oraz przycisk do zamknięcia powiększonego obrazu. Skrypt gallery.js odpowiada za logikę pobierania zdjęć z API NASA, wyświetlanie ich w galerii oraz obsługę powiększania obrazów po kliknięciu. Struktura strony jest prosta i intuicyjna, zapewniając użytkownikom wygodne przeglądanie zdjęć. Galeria jest w pełni zintegrowana z aplikacją, umożliwiając interakcję z danymi wizualnymi dostarczonymi przez NASA.

```

frontend > templates > <> gallery.html > ...
{% load static %}
<!DOCTYPE html>
<html lang="pl">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Galeria NASA</title>
    <link rel="stylesheet" href="{% static 'styles/gallery.css' %}" />
  </head>
  <body>
    <nav id="navbar">
      <div id="app-name">Mars-Dust</div>
      <div id="auth-buttons">
        <a href="{% url 'home' %}"><button>Strona Główna</button></a>
      </div>
    </nav>

    <main id="gallery-section">
      <h1>Galeria z API NASA</h1>
      <div id="gallery-container">
        <p id="loading-text">Ładowanie zdjęć...</p>
      </div>
    </main>

    <div id="lightbox" class="hidden">
      <div id="lightbox-content">
        <img id="lightbox-img" src="" alt="Powiększone zdjęcie" />
        <button id="close-lightbox">X</button>
      </div>
    </div>

    <script src="{% static 'js/gallery.js' %}"></script>
  </body>
</html>

```

## Plik main.js

Plik `main.js` jest kluczowym komponentem aplikacji Mars-Dust, odpowiedzialnym za renderowanie modelu 3D Marsa, jego księżyców Phobos i Deimos oraz obsługę interakcji użytkownika. Skrypt wykorzystuje bibliotekę Three.js do tworzenia sceny 3D, w której znajduje się model Marsa z teksturami oraz księżyce orbitujące wokół niego. Scena jest oświetlana za pomocą światła otoczenia i kierunkowego, co nadaje realistyczny wygląd. Dodatkowo, w tle generowane są gwiazdy, które tworzą efekt kosmicznego otoczenia. Interakcje użytkownika obejmują możliwość obracania Marsa za pomocą myszy, powiększania i pomniejszania widoku przy użyciu scrolla oraz przełączania między sektorami na powierzchni planety. Każdy sektor ma przypisane informacje, które są wyświetlane w panelu informacyjnym po kliknięciu odpowiedniego przycisku. Aplikacja obsługuje również panele logowania i rejestracji, które komunikują się z backendem przez API, wykorzystując tokeny CSRF do bezpiecznej autentykacji. Skrypt zawiera funkcje do pobierania danych o sektorach, burzach piaskowych, instalacjach oraz innych elementach związanych z eksploracją Marsa. Dane są dynamicznie aktualizowane i wyświetlane w panelach informacyjnych. Dodatkowo, aplikacja integruje się z zewnętrznym API, aby pobierać informacje o księżycach Phobos i Deimos, takie jak ich rozmiar, orbita czy odkrywca. Plik `main.js` jest modularny i dobrze zorganizowany, co ułatwia rozbudowę i utrzymanie kodu. Skrypt obsługuje również responsywność, dostosowując rozmiar sceny 3D do wymiarów okna przeglądarki.

```

const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.getElementById('mars-container').appendChild(renderer.domElement);

// Wczytanie tekstury Marsa
const textureLoader = new THREE.TextureLoader();
const marsTexture = textureLoader.load(window.imagePath);
const geometry = new THREE.SphereGeometry(5, 64, 64);
const material = new THREE.MeshStandardMaterial({
  map: marsTexture,
  metalness: 0.4,
  roughness: 0.6,
});
const mars = new THREE.Mesh(geometry, material);
scene.add(mars);

// Oświetlenie sceny
const ambientLight = new THREE.AmbientLight(0x404040);
const directionalLight = new THREE.DirectionalLight(0xffffff, 1);
directionalLight.position.set(10, 10, 10);
scene.add(ambientLight);
scene.add(directionalLight);

// Kamera
camera.position.z = 15;

// Gwiazdy w tle
const starsGeometry = new THREE.BufferGeometry();
const starsMaterial = new THREE.PointsMaterial({ color: 0x888888, size: 0.05 });
const starPositions = Array.from({ length: 10000 }, () => (Math.random() - 0.5) * 2000);
starsGeometry.setAttribute('position', new THREE.Float32BufferAttribute(starPositions, 3));
const stars = new THREE.Points(starsGeometry, starsMaterial);
scene.add(stars);

// Księżyc Marsa: Fobos i Deimos
const moons = [];

function createMoon(size, distance, speed, texturePath, name) {
  const moonGeometry = new THREE.SphereGeometry(size, 32, 32);
  const moonMaterial = new THREE.MeshStandardMaterial({
    map: textureLoader.load(texturePath),
    metalness: 0.3,
    roughness: 0.8
  });
  const moon = new THREE.Mesh(moonGeometry, moonMaterial);
  moon.position.set(distance, 0, 0);
  moon.name = name;
  mars.add(moon);

  moons.push({
    mesh: moon,
    distance: distance,
    speed: speed,
    angle: Math.random() * Math.PI * 2
  });
}

createMoon(0.9, 11, 0.015, window.fobosTexture, 'Phobos');
createMoon(0.5, 16, 0.01, window.deimosTexture, 'Deimos');

```

```

// Animacja
function animate() {
  requestAnimationFrame(animate);
  moons.forEach(moon => {
    moon.angle += moon.speed;
    moon.mesh.position.x = Math.cos(moon.angle) * moon.distance;
    moon.mesh.position.z = Math.sin(moon.angle) * moon.distance;
  });
  mars.rotation.y += 0.005;
  renderer.render(scene, camera);
}
animate();

// Tu jest interakcja z planetą
let isMouseDown = false;
let previousMousePosition = { x: 0, y: 0 };

document.addEventListener('mousedown', () => (isMouseDown = true));
document.addEventListener('mouseup', () => (isMouseDown = false));
document.addEventListener('mousemove', (e) => {
  if (isMouseDown) {
    mars.rotation.y += (e.clientX - previousMousePosition.x) * 0.005;
    mars.rotation.x += (e.clientY - previousMousePosition.y) * 0.005;
  }
  previousMousePosition = { x: e.clientX, y: e.clientY };
});

document.addEventListener('wheel', (e) => {
  camera.position.z += e.deltaY * 0.01;
});

window.addEventListener('resize', () => {
  renderer.setSize(window.innerWidth, window.innerHeight);
  camera.aspect = window.innerWidth / window.innerHeight;
  camera.updateProjectionMatrix();
});

// Panel z sektorami
const sectorButtonsContainer = document.getElementById('sector-buttons');
const prevButton = document.getElementById('prev-sector');
const nextButton = document.getElementById('next-sector');

const totalSectors = 20;
const buttonsPerPage = 5;
let currentPage = 0;

function generateSectorButtons() {
  sectorButtonsContainer.innerHTML = '';

  for (let i = currentPage * buttonsPerPage + 1; i <= (currentPage + 1) * buttonsPerPage && i <= totalSectors; i++) {
    const button = document.createElement('button');
    button.innerText = `Sektor ${i}`;
    button.addEventListener('click', () => showSectorInfo(i));
    sectorButtonsContainer.appendChild(button);
  }
}

```



```

// Panel z informacjami o sektorze
let currentSector = null;

// Funkcja wyświetlająca informacje o sektorze
function showSectorInfo(sector) {
    if (!loggedIn == false) {
        const infoText = document.getElementById('info-text-logout');
        const details = [
            1: "Sektor 1: Prognoza burz na najbliższy tydzień jest spokojna.",
            2: "Sektor 2: Ostrzeżenie o możliwych burzach piaskowych.",
            3: "Sektor 3: Stabilne warunki pogodowe, idealne do eksploracji.",
            4: "Sektor 4: Wysoka aktywność pyłowa, zachowaj ostrożność.",
            5: "Sektor 5: Optymalne warunki dla misji badawczej.",
            6: "Sektor 6: Średnie ryzyko burz piaskowych, zaleca się ograniczenie działań na otwartej przestrzeni.",
            7: "Sektor 7: Warunki pogodowe stabilne, ale możliwa podwyższona aktywność pyłowa w godzinach wieczornych.",
            8: "Sektor 8: Niska widoczność z powodu unoszącego się pyłu, konieczne stosowanie dodatkowych filtrów ochronnych.",
            9: "Sektor 9: Spokojna pogoda, idealna do konserwacji instalacji i sprzętu.",
            10: "Sektor 10: Silne wiatry przewidywane na popołudnie, możliwe opóźnienia w komunikacji terenowej.",
            11: "Sektor 11: Warunki sprzyjające budowie bazy, przewidywana stabilność atmosferyczna.",
            12: "Sektor 12: Stabilne warunki atmosferyczne, sprzyjające testom nowych technologii.",
            13: "Sektor 13: Podwyższona aktywność burz pyłowych w rejonach południowych, sugierana ostrożność.",
            14: "Sektor 14: Niemedkie podmuchy wiatru i umiarkowane warunki atmosferyczne, dobre do misji zwiadowczych.",
            15: "Sektor 15: Możliwe krótkotrwałe burze magnetyczne, zachowaj ostrożność przy obsłudze urządzeń elektronicznych.",
            16: "Sektor 16: Warunki atmosferyczne sprzyjające eksploracji, minimalna aktywność pyłowa.",
            17: "Sektor 17: Średnie ryzyko burz, zaleca się monitorowanie warunków w czasie rzeczywistym.",
            18: "Sektor 18: Spokojne warunki atmosferyczne, idealne do instalacji nowych paneli solarnych.",
            19: "Sektor 19: Lekka aktywność pyłowa, zalecane działania prewencyjne na systemach wentylacyjnych.",
            20: "Sektor 20: Przewidywana stabilność atmosferyczna, doskonała okazja do rozszerzenia działań naukowych."
        ];
        infoText.innerText = details[sector] || "Brak danych o wybranym sektorze.";
        currentSector = sector;
        document.getElementById('info-panel-logout').style.left = '0px';
    }
    else {
        fetchSectorDetails(sector);
    }
}

// Funkcja wyświetlająca informacje o bazie
function showBaseInfo() {
    const baseInfo = [
        1: "Baza w sektorze 1: Oczekiwana stabilność atmosferyczna, idealna do dalszej eksploracji.",
        2: "Baza w sektorze 2: Należy zachować ostrożność, burze piaskowe mogą występować.",
        3: "Baza w sektorze 3: Idealne warunki do długoterminowego pobytu.",
        4: "Baza w sektorze 4: Zbudowanie bazy jest ryzykowne z powodu dużej aktywności pyłowej.",
        5: "Baza w sektorze 5: Stabilne warunki, bezpieczne dla długoterminowego osiedlenia.",
        6: "Baza w sektorze 6: Warunki umiarkowane, ale możliwe krótkotrwałe burze piaskowe [ ] zaleca się ostrożność przy budowie.",
        7: "Baza w sektorze 7: Stabilna pogoda w większości rejonów, ale lokalna aktywność pyłowa może wpływać na widoczność.",
        8: "Baza w sektorze 8: Niska widoczność z powodu unoszącego się pyłu, ale stabilne warunki do instalacji sprzętu ochronnego.",
        9: "Baza w sektorze 9: Spokojne warunki atmosferyczne sprzyjają długoterminowym operacjom technicznym.",
        10: "Baza w sektorze 10: Możliwe okresowe silne wiatry, zaleca się zabezpieczenie struktur budowlanych.",
        11: "Baza w sektorze 11: Warunki sprzyjające budowie bazy, przewidywana stabilność atmosferyczna.",
        12: "Baza w sektorze 12: Stabilne warunki pogodowe, idealne do rozwijania infrastruktury naukowej.",
        13: "Baza w sektorze 13: Podwyższone ryzyko burz pyłowych w niektórych rejonach [ ] sugerowane wzmocnienie systemów ochronnych.",
        14: "Baza w sektorze 14: Umiarkowane warunki atmosferyczne, dobre do rozpoczęcia mniejszych projektów osadniczych.",
        15: "Baza w sektorze 15: Możliwe zakłócenia magnetyczne, ale ogólnie bezpieczne warunki do budowy i eksploatacji.",
        16: "Baza w sektorze 16: Stabilne warunki atmosferyczne sprzyjają rozwojowi nowych instalacji technicznych.",
        17: "Baza w sektorze 17: Średnie ryzyko wystąpienia burz pyłowych [ ] zalecana ciągła obserwacja pogody.",
        18: "Baza w sektorze 18: Stabilna pogoda idealna do robót przy paneli solarnych i infrastruktury energetycznej.",
        19: "Baza w sektorze 19: Aktywność pyłowa jest minimalna, ale zalecane są prewencyjne środki ochrony sprzętu.",
        20: "Baza w sektorze 20: Warunki atmosferyczne są bardzo korzystne, idealne miejsce na dalszą ekspansję naukową."
    ];
}

```

```

// Obsługa kliknięcia w przycisk "Baza"
document.getElementById('base-button').addEventListener('click', showBaseInfo);

// Inne przyciski (np. "Zamknij")
document.getElementById('close-panel-login').addEventListener('click', () => {
    document.getElementById('info-panel-login').style.left = '-400px';
});

document.getElementById('close-panel-logout').addEventListener('click', () => {
    document.getElementById('info-panel-logout').style.left = '-400px';
});

// Obsługa przycisków nawigacyjnych
prevButton.addEventListener('click', () => {
    if (currentPage > 0) {
        currentPage--;
        generateSectorButtons();
    }
});

nextButton.addEventListener('click', () => {
    if ((currentPage + 1) * buttonsPerPage < totalSectors) {
        currentPage++;
        generateSectorButtons();
    }
});

async function fetchSectorDetails(sectorId) {
    try {
        const response = await fetch(`/api/sectors/${sectorId}/`);
        if (!response.ok) {
            throw new Error("Nie udało się pobrać danych o sektorze.");
        }
        const data = await response.json();
        updateSectorInfo(data); // Funkcja do aktualizacji panelu informacyjnego
    } catch (error) {
        console.error(error);
        alert("Nie udało się pobrać danych o sektorze.");
    }
}

function updateSectorInfo(data) {
    const infoText = document.getElementById('info-text-login');

    const installationLink = data["Installation"] !== "Brak instalacji"
    ? `<a href="#" onclick="fetchInstallationDetails('${data["Installation ID"]}'); return false;">${data["Installation"]}</a>`
    : "Brak instalacji";

    infoText.innerHTML = `
        <h3>Sektor ${data["Info"]}</h3>
        <p>Nazwa sektora: <strong> ${data["Sector name"]} </strong></p>
        <p>Opis: <strong> ${data["Description"]} </strong></p>
        <p>Minimalna szerokość geograficzna: <strong> ${data["Min latitude"]} </strong></p>
        <p>Maksymalna szerokość geograficzna: <strong> ${data["Max latitude"]} </strong></p>
        <p>Minimalna długość geograficzna: <strong> ${data["Min longitude"]} </strong></p>
        <p>Maksymalna długość geograficzna: <strong> ${data["Max longitude"]} </strong></p>
        <p>Instalacja: <strong> ${installationLink}</p>
    `;
    document.getElementById('info-panel-login').style.left = '0px';
}

```

```

async function fetchStormDetails(storm_id) {
  try {
    const response = await fetch(`/api/storms/${storm_id}/`);
    if (!response.ok) {
      throw new Error("Nie udało się pobrać danych o burzy.");
    }
    const data = await response.json();
    updateStormInfo(data); // Funkcja do aktualizacji panelu informacyjnego
  } catch (error) {
    console.error(error);
    alert("Nie udało się pobrać danych o burzy.");
  }
}

function updateStormInfo(data) {
  const damageLink = data["Damage"] ? "Brak uszkodzeń"
  : `<a href="#" onclick="fetchDamageDetails('${data["Damage ID"]}'); return false;">${data["Damage"]}</a>`
  : "Brak uszkodzeń";

  const infoText = document.getElementById("info-text-login");
  infoText.innerHTML = `
    <h3>Burza ${data["Info"]}</h3>
    <p><strong>Szerokość geograficzna centroidy:</strong> ${data["Centroid latitude"]}</p>
    <p><strong>Długość geograficzna centroidy:</strong> ${data["Centroid longitude"]}</p>
    <p><strong>Czas trwania:</strong> ${data["Duration"]}</p>
    <p><strong>Rok marsjański:</strong> ${data["Mars year"]}</p>
    <p><strong>ID fazy:</strong> ${data["Member ID"]}</p>
    <p><strong>Subfaza misji:</strong> ${data["Mission subphase"]}</p>
    <p><strong>Siła:</strong> ${data["Power"]}</p>
    <p><strong>Słońce:</strong> ${data["Sol"]}</p>
    <p><strong>Szerokość burzy:</strong> ${data["Spread latitude"]}</p>
    <p><strong>Długość burzy:</strong> ${data["Spread longitude"]}</p>
    <p><strong>Uszkodzenia od kamieni:</strong> ${data["Stone damage"]}</p>
    <p><strong>Uszkodzenie:</strong> ${damageLink}</p>
  `;
  document.getElementById("info-panel-login").style.left = "0px";
}

async function fetchSpecialityDetails(speciality_id) {
  try {
    const response = await fetch(`/api/specialities/${speciality_id}/`);
    if (!response.ok) {
      throw new Error("Nie udało się pobrać danych o specjalizacji.");
    }
    const data = await response.json();
    updateSpecialityInfo(data); // Funkcja do aktualizacji panelu informacyjnego
  } catch (error) {
    console.error(error);
    alert("Nie udało się pobrać danych o specjalizacji.");
  }
}

function updateSpecialityInfo(data) {
  const staffLink = data["Staff"] ? "Brak personelu"
  : `<a href="#" onclick="fetchStaffDetails('${data["Staff ID"]}'); return false;">${data["Staff"]}</a>`
  : "Brak personelu";

  const infoText = document.getElementById("info-text-login");
  infoText.innerHTML = `
    <h3>Specjalizacja ${data["Info"]}</h3>
    <p><strong>Nazwa:</strong> ${data["Name"]}</p>
    <p><strong>Pracownik:</strong> ${staffLink}</p>
  `;
  document.getElementById("info-panel-login").style.left = "0px";
}

```

```

async function fetchStaffDetails(staff_id) {
  try {
    const response = await fetch(`/api/staff/${staff_id}/`);
    if (!response.ok) {
      throw new Error("Nie udało się pobrać danych o pracowniku.");
    }
    const data = await response.json();
    updateStaffInfo(data); // Funkcja do aktualizacji panelu informacyjnego
  } catch (error) {
    console.error(error);
    alert("Nie udało się pobrać danych o pracowniku.");
  }
}

function updateStaffInfo(data) {
  const specialityLink = data["Speciality"] ? "Brak specjalizacji"
  : `<a href="#" onclick="fetchSpecialityDetails('${data["Speciality ID"]}'); return false;">${data["Speciality"]}</a>`
  : "Brak specjalizacji";

  const conservationScheduleLink = data["Conservation schedule"] ? "Brak napraw"
  : `<a href="#" onclick="fetchConservationScheduleDetails('${data["Conservation schedule ID"]}'); return false;">${data["Conservation schedule"]}</a>`
  : "Brak napraw";

  const infoText = document.getElementById("info-text-login");
  infoText.innerHTML = `
    <h3>Pracownik ${data["Info"]}</h3>
    <p><strong>Specjalizacja:</strong> ${specialityLink}</p>
    <p><strong>Imię:</strong> ${data["Name"]}</p>
    <p><strong>Nazwisko:</strong> ${data["Surname"]}</p>
    <p><strong>Cechy:</strong> ${data["Traits"]}</p>
    <p><strong>Naprawa:</strong> ${conservationScheduleLink}</p>
  `;
  document.getElementById("info-panel-login").style.left = "0px";
}

async function fetchConservationScheduleDetails(task_id) {
  try {
    const response = await fetch(`/api/conservation_schedules/${task_id}/`);
    if (!response.ok) {
      throw new Error("Nie udało się pobrać danych o naprawie.");
    }
    const data = await response.json();
    updateConservationScheduleInfo(data); // Funkcja do aktualizacji panelu informacyjnego
  } catch (error) {
    console.error(error);
    alert("Nie udało się pobrać danych o naprawie.");
  }
}

```

```

async function fetchPartDetails(part_id) {
  try {
    const response = await fetch(`/api/parts/${part_id}/`);
    if (!response.ok) {
      throw new Error("Nie udało się pobrać danych o części.");
    }
    const data = await response.json();
    updatePartInfo(data); // Funkcja do aktualizacji panelu informacyjnego
  } catch (error) {
    console.error(error);
    alert("Nie udało się pobrać danych o części.");
  }
}

function updatePartInfo(data) {
  const installationLink = data["Installation"] !== "Brak instalacji"
    ? `<a href="#" onclick="fetchInstallationDetails(${data["Installation ID"]}); return false;">${data["Installation"]}</a>`
    : "Brak instalacji";

  const partUsageLink = data["Part Usage"] !== "Brak użycia części"
    ? `<a href="#" onclick="fetchPartUsageDetails(${data["Part Usage ID"]}); return false;">${data["Part Usage"]}</a>`
    : "Brak użycia części";

  const damageLink = data["Damage"] !== "Brak uszkodzeń"
    ? `<a href="#" onclick="fetchDamageDetails(${data["Damage ID"]}); return false;">${data["Damage"]}</a>`
    : "Brak uszkodzeń";

  const infoText = document.getElementById("info-text-login");
  infoText.innerHTML = `
    <h3>Część ${data["Info"]}</h3>
    <p><strong>Instalacja:</strong> ${installationLink}</p>
    <p><strong>Nazwa:</strong> ${data["Name"]}</p>
    <p><strong>Użycie części:</strong> ${partUsageLink}</p>
    <p><strong>Uszkodzenie:</strong> ${damageLink}</p>
  `;
  document.getElementById("info-panel-login").style.left = "0px";
}

async function fetchInstallationDetails(installation_id) {
  try {
    const response = await fetch(`/api/installations/${installation_id}/`);
    if (!response.ok) {
      throw new Error("Nie udało się pobrać danych o instalacji.");
    }
    const data = await response.json();
    updateInstallationInfo(data); // Funkcja do aktualizacji panelu informacyjnego
  } catch (error) {
    console.error(error);
    alert("Nie udało się pobrać danych o instalacji.");
  }
}

function updateInstallationInfo(data) {
  const sectorLink = data["Sector Usage"] !== "Brak sektora"
    ? `<a href="#" onclick="fetchSectorDetails(${data["Sector ID"]}); return false;">${data["Sector"]}</a>`
    : "Brak sektora";

  const partUsageLink = data["Part usage"] !== "Brak użycia części"
    ? `<a href="#" onclick="fetchPartUsageDetails(${data["Part usage ID"]}); return false;">${data["Part usage"]}</a>`
    : "Brak użycia części";

  const infoText = document.getElementById("info-text-login");
  infoText.innerHTML = `
    <h3>Instalacja ${data["Info"]}</h3>
  `;

```

```

    <h3>Instalacja ${data["Info"]}</h3>
    <p><strong>Sektor:</strong> ${sectorLink}</p>
    <p><strong>Nazwa:</strong> ${data["Name"]}</p>
    <p><strong>Użycie części:</strong> ${partUsageLink}</p>
  `;
  document.getElementById("info-panel-login").style.left = "0px";
}

async function fetchPartUsageDetails(part_usage_id) {
  try {
    const response = await fetch(`/api/parts_usages/${part_usage_id}/`);
    if (!response.ok) {
      throw new Error("Nie udało się pobrać danych o użyciu części.");
    }
    const data = await response.json();
    updatePartUsageInfo(data); // Funkcja do aktualizacji panelu informacyjnego
  } catch (error) {
    console.error(error);
    alert("Nie udało się pobrać danych o użyciu części.");
  }
}

function updatePartUsageInfo(data) {
  const partLink = data["Part"] !== "Brak części"
    ? `<a href="#" onclick="fetchPartDetails(${data["Part ID"]}); return false;">${data["Part"]}</a>`
    : "Brak części";

  const installationLink = data["Installation"] !== "Brak instalacji"
    ? `<a href="#" onclick="fetchInstallationDetails(${data["Installation ID"]}); return false;">${data["Installation"]}</a>`
    : "Brak instalacji";

  const infoText = document.getElementById("info-text-login");
  infoText.innerHTML = `
    <h3>Użycie części ${data["Info"]}</h3>
    <p><strong>Część:</strong> ${partLink}</p>
    <p><strong>Instalacja:</strong> ${installationLink}</p>
  `;
  document.getElementById("info-panel-login").style.left = "0px";
}

async function fetchDamageDetails(damage_id) {
  try {
    const response = await fetch(`/api/damages/${damage_id}/`);
    if (!response.ok) {
      throw new Error("Nie udało się pobrać danych o uszkodzeniu.");
    }
    const data = await response.json();
    updateDamageInfo(data); // Funkcja do aktualizacji panelu informacyjnego
  } catch (error) {
    console.error(error);
    alert("Nie udało się pobrać danych o użyciu uszkodzeniu.");
  }
}

function updateDamageInfo(data) {
  const stormLink = data["Cause"] !== "Brak burzy"
    ? `<a href="#" onclick="fetchStormDetails(${data["Cause ID"]}); return false;">${data["Cause"]}</a>`
    : "Brak burzy";

  const partLink = data["Part"] !== "Brak części"
    ? `<a href="#" onclick="fetchPartDetails(${data["Part ID"]}); return false;">${data["Part"]}</a>`

```

```

function updateDamageInfo(data) {
  const conservationScheduleLink = data["Queued task"] !== "Brak napraw"
  ? <a href="#" onclick="fetchConservationScheduleDetails(${data["Queued task ID"]}); return false;">${data["Queued task"]}</a>
  : "Brak napraw";

  const infoText = document.getElementById("info-text-login");
  infoText.innerHTML = `
    <div>Zobaczenie ${data["Info"]}</div></div>
    <p>${strong>Podejście</strong> ${stateLink}</p>
    <p>${strong>Część</strong> ${partLink}</p>
    <p>${strong>Zaplanowana naprawa</strong> ${conservationScheduleLink}</p>
    <p>${strong>Szczegółowa czy podejmowana</strong> ${data["Presumed or reported"]}</p>
    <p>${strong>Poważność</strong> ${data["Severity"]}</p>
  `;
  document.getElementById("info-panel-login").style.left = "8px";
}

const closePanelLogoutButton = document.getElementById("close-panel-logout");
const closePanelLoginButton = document.getElementById("close-panel-login");

closePanelLoginButton.addEventListener('click', () => {
  document.getElementById("info-panel-login").style.left = "-400px";
});

closePanelLogoutButton.addEventListener('click', () => {
  document.getElementById("info-panel-logout").style.left = "-400px";
});

generateSectorButtons();

// Panel logowania i rejestracji
const loginPanel = document.getElementById("login-panel");
const registerPanel = document.getElementById("register-panel");
const loginButton = document.getElementById("login-button");
const registerButton = document.getElementById("register-button");
const closeLoginPanelButton = document.getElementById("close-login-panel");
const adminPanelButton = document.getElementById("admin-panel-button");
const closeRegisterPanelButton = document.getElementById("close-register-panel");
let isLoggedIn = false;

loginButton.addEventListener('click', () => {
  loginPanel.style.right = "8px";
  registerPanel.style.right = "-400px";
});

registerButton.addEventListener('click', () => {
  registerPanel.style.right = "8px";
  loginPanel.style.right = "-400px";
});

closeLoginPanelButton.addEventListener('click', () => {
  loginPanel.style.right = "-400px";
});

closeRegisterPanelButton.addEventListener('click', () => {
  registerPanel.style.right = "-400px";
});

adminPanelButton.addEventListener('click', () => {
  document.location.href = "http://127.0.0.1:8080/admin/login/?next=/admin/";
});

```

```

// URLs for API endpoints
const API_LOGIN_URL = "/api/login/";
const API_LOGOUT_URL = "/api/logout/";
const API_REGISTER_URL = "/api/register/";

// Utility to send POST requests
async function postData(url = "", data = {}) {
  const response = await fetch(url, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      "X-CSRFToken": csrfToken,
    },
    body: JSON.stringify(data),
  });

  if (!response.ok) {
    const errorData = await response.json();
    throw new Error(errorData.error || "An error occurred.");
  }

  return response.json();
}

// Handle login
async function handleLogin(event) {
  event.preventDefault();
  const username = document.querySelector("#login-panel input[type='text']").value;
  const password = document.querySelector("#login-panel input[type='password']").value;

  try {
    const result = await postData(API_LOGIN_URL, { username, password });
    alert("Logowanie udane!");
    document.querySelector("#login-panel").style.right = "-400px"; // Close the login panel
    document.querySelector("#auth-buttons").innerHTML = `
      <button id="logout-button">Wyloguj</button>
    `;
    document.querySelector("#logout-button").addEventListener("click", handleLogout);
    isLoggedIn = true;
  } catch (error) {
    alert(error.message);
  }
}

// Handle logout
async function handleLogout() {
  try {
    await postData(API_LOGOUT_URL, {});
    alert("Wylogowanie udane!");
    window.location.href = "/"; // Przekierowanie na stronę główną
    isLoggedIn = false;
  } catch (error) {
    alert(error.message);
  }
}

// Handle registration
async function handleRegister(event) {
  event.preventDefault();
  const username = document.querySelector("#register-panel input[type='text']").value;
  const email = document.querySelector("#register-panel input[type='email']").value;
  const password = document.querySelector("#register-panel input[type='password']").value;
  const confirmPassword = document.querySelectorAll("#register-panel input[type='password']")[1].value;

  try {
    const result = await postData(API_REGISTER_URL, { username, email, password, confirmPassword });
  }

```

```

    try {
      const result = await postData(API_REGISTER_URL, { username, email, password, confirm_password: confirmPassword });
      alert("Rejestracja udana! Proszę się zalogować.");
      document.querySelector("#register-panel").style.right = "-400px"; // Close the register panel
    } catch (error) {
      alert(error.message);
    }
  }

  // Attach event listeners
  document.querySelector("#login-panel form").addEventListener("submit", handleLogin);
  document.querySelector("#register-panel form").addEventListener("submit", handleRegister);

  // Attach event listeners
  document.querySelector("#login-button").addEventListener("click", () => {
    document.querySelector("#login-panel").style.right = "0";
  });

  document.querySelector("#close-login-panel").addEventListener("click", () => {
    document.querySelector("#login-panel").style.right = "-400px";
  });

  document.getElementById('phobos-button').addEventListener('click', () => {
    document.getElementById('phobos-panel').style.left = '0px';
  });

  document.getElementById('deimos-button').addEventListener('click', () => {
    document.getElementById('deimos-panel').style.left = '0px';
  });

  document.getElementById('close-phobos-panel').addEventListener('click', () => {
    document.getElementById('phobos-panel').style.left = "-400px";
  });

  document.getElementById('close-deimos-panel').addEventListener('click', () => {
    document.getElementById('deimos-panel').style.left = "-400px";
  });

  const apiKey = '165f5e2d1198787af9d0406e5d40c37d0dc';
  const phobosDataUrl = 'https://api.le-systeme-solaire.net/rest/bodies/phobos?apiKey=${apiKey}';
  const deimosDataUrl = 'https://api.le-systeme-solaire.net/rest/bodies/deimos?apiKey=${apiKey}';

  async function fetchMoonData(moonUrl) {
    const response = await fetch(moonUrl);
    const data = await response.json();
    return data;
  }

  async function displayMoonInfo() {
    try {
      const phobosData = await fetchMoonData(phobosDataUrl);
      const deimosData = await fetchMoonData(deimosDataUrl);

      const phobosInfo = `
        <h3>Phobos</h3>
        <p><strong>Wielkość:</strong> ${phobosData.mass.massValue * Math.pow(10, phobosData.mass.massExponent)} kg</p>
        <p><strong>Czas obiegu:</strong> ${phobosData.sideralOrbit} dni</p>
        <p><strong>Średnica:</strong> ${phobosData.meanRadius * 2} km</p>
        <p><strong>Typ orbity:</strong> Elipsoidalna (Eccentricity: ${phobosData.eccentricity})</p>
        <p><strong>Odległość od Marsa:</strong> Średnia: ${phobosData.semimajorAxis} km, Perihelion: ${phobosData.perihelion} km, Aphelion: ${phobosData.apelion} km</p>
        <p><strong>Temperatura powierzchni:</strong> Brak danych</p>
        <p><strong>Okres rotacji:</strong> ${phobosData.sideralRotation} godzin</p>
        <p><strong>Kompozycja chemiczna:</strong> Brak danych</p>
        <p><strong>Wielkość (wymiar):</strong> ${phobosData.dimension}</p>
        <p><strong>Grawitacja:</strong> ${phobosData.gravity} m/s</p>
        <p><strong>Escape Velocity:</strong> ${phobosData.escape} m/s</p>
        <p><strong>Odkrywcą:</strong> ${phobosData.discoveredBy}</p>
        <p><strong>Data odkrycia:</strong> ${phobosData.discoveryDate}</p>
      `;

      const deimosInfo = `
        <h3>Deimos</h3>
        <p><strong>Wielkość:</strong> ${deimosData.mass.massValue * Math.pow(10, deimosData.mass.massExponent)} kg</p>
        <p><strong>Czas obiegu:</strong> ${deimosData.sideralOrbit} dni</p>
        <p><strong>Średnica:</strong> ${deimosData.meanRadius * 2} km</p>
        <p><strong>Typ orbity:</strong> Prawie kołowa (Eccentricity: ${deimosData.eccentricity})</p>
        <p><strong>Odległość od Marsa:</strong> Średnia: ${deimosData.semimajorAxis} km, Perihelion: ${deimosData.perihelion} km, Aphelion: ${deimosData.apelion} km</p>
        <p><strong>Temperatura powierzchni:</strong> Brak danych</p>
        <p><strong>Okres rotacji:</strong> ${deimosData.sideralRotation} godzin</p>
        <p><strong>Kompozycja chemiczna:</strong> Brak danych</p>
        <p><strong>Wielkość (wymiar):</strong> ${deimosData.dimension}</p>
        <p><strong>Grawitacja:</strong> ${deimosData.gravity} m/s</p>
        <p><strong>Escape Velocity:</strong> ${deimosData.escape} m/s</p>
        <p><strong>Odkrywcą:</strong> ${deimosData.discoveredBy}</p>
        <p><strong>Data odkrycia:</strong> ${deimosData.discoveryDate}</p>
      `;

      document.getElementById('phobos-text').innerHTML = phobosInfo;
      document.getElementById('deimos-text').innerHTML = deimosInfo;
    } catch (error) {
      console.error("Error fetching moon data:", error);
    }
  }

  // Wywołaj funkcję do załadowania danych po załadowaniu strony
  window.addEventListener('load', displayMoonInfo);

```

## Plik gallery.js

Plik gallery.js odpowiada za dynamiczne ładowanie zdjęć z API NASA i wyświetlanie ich w galerii na stronie galery.html. Po załadowaniu strony, skrypt rozpoczyna pobieranie danych z API NASA dotyczących zdjęć związanych z Marsem, korzystając z URL: [https://images-api.nasa.gov/search?q=mars&media\\_type=image](https://images-api.nasa.gov/search?q=mars&media_type=image). Po uzyskaniu danych, skrypt sprawdza, czy w odpowiedzi znajdują się jakiekolwiek zdjęcia i w przypadku ich braku wyświetla komunikat o braku wyników. W przeciwnym razie, ukrywa tekst ładowania i zaczyna tworzyć elementy galerii. Każde zdjęcie jest reprezentowane jako element div z tłem ustawionym na obraz z API oraz tytułem pobranym z danych. Elementy galerii są dynamicznie dodawane do kontenera gallery-container. Po kliknięciu na obraz, skrypt wyświetla powiększoną wersję zdjęcia w sekcji lightbox. Użytkownik może zamknąć powiększone zdjęcie, klikając przycisk "X", co powoduje ukrycie sekcji lightbox. Skrypt obsługuje również błędy, takie jak problem z pobieraniem danych z API, wyświetlając odpowiedni komunikat w przypadku niepowodzenia. Dzięki temu użytkownik zawsze otrzymuje informację o stanie procesu ładowania zdjęć lub wystąpieniu problemów. Plik ten jest kluczowy dla funkcjonalności galerii, zapewniając interaktywne i płynne przeglądanie zdjęć NASA na stronie.

```
document.addEventListener("DOMContentLoaded", () => {
  const API_URL = "https://images-api.nasa.gov/search?q=mars&media_type=image";
  const galleryContainer = document.getElementById("gallery-container");
  const lightbox = document.getElementById("lightbox");
  const lightboxImg = document.getElementById("lightbox-img");
  const closeLightbox = document.getElementById("close-lightbox");
  const loadingText = document.getElementById("loading-text");

  const loadPhotos = async () => {
    try {
      const response = await fetch(API_URL);
      const data = await response.json();

      if (data.collection && data.collection.items) {
        const items = data.collection.items;
        if (items.length === 0) {
          galleryContainer.innerHTML = "<p>Brak wyników związanych z Marsem.</p>";
          return;
        }

        loadingText.style.display = "none";
        galleryContainer.innerHTML = "";

        items.forEach(item => {
          if (item.links && item.links[0]) {
            const mediaUrl = item.links[0].href;
            const mediaTitle = item.data[0].title || "Unknown Title";

            const imgElement = document.createElement("div");
            imgElement.classList.add("gallery-item");
            imgElement.style.backgroundImage = `url('${mediaUrl}')`;
            imgElement.title = mediaTitle;

            imgElement.addEventListener("click", () => {
              lightboxImg.src = mediaUrl;
              lightboxImg.alt = mediaTitle;
              lightbox.classList.remove("hidden");
            });

            galleryContainer.appendChild(imgElement);
          }
        });
      } else {
        galleryContainer.innerHTML = "<p>Brak wyników związanych z Marsem.</p>";
      }
    } catch (error) {
      console.error("Error loading photos:", error);
      galleryContainer.innerHTML = "<p>Błąd wczytywania danych.</p>";
    }
  };

  loadPhotos();

  closeLightbox.addEventListener("click", () => {
    lightbox.classList.add("hidden");
    lightboxImg.src = "";
  });
});
```

## Plik urls.py

Plik urls.py jest odpowiedzialny za definiowanie tras URL aplikacji w Django. Określa, które widoki mają być wywoływane dla danych ścieżek URL w aplikacji Mars-Dust. W tym przypadku, na początku znajduje się definicja kilku tras, które prowadzą do widoków takich jak home, admin\_dashboard, gallery, information. Ponadto, są również zdefiniowane ścieżki dla API, które obsługują logowanie, rejestrację użytkowników, a także dostęp do szczegółowych danych o burzach piaskowych, sektorach, specjalizacjach, pracownikach, harmonogramach konserwacji, częściach, instalacjach, użyciach części oraz uszkodzeniach. Wszystkie te ścieżki API przyjmują odpowiednie identyfikatory w URL, co pozwala na dynamiczne pobieranie szczegółów związanych z poszczególnymi obiektami. Plik ten jest kluczowy dla nawigacji w aplikacji oraz dla udostępniania danych w formacie API, umożliwiając interakcję z aplikacją zarówno za pomocą tradycyjnych stron, jak i zapytań AJAX czy innych metod dostępu do danych.

```
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home, name='home'),
    path('admin.html', views.admin_dashboard, name='admin_dashboard'),
    path('gallery.html', views.gallery, name='gallery'),
    path('info.html', views.information, name='information'),
    path('api/login/', views.LoginView.as_view(), name='api_login'),
    path('api/logout/', views.LogoutView.as_view(), name='api_logout'),
    path('api/register/', views.RegisterAPIView.as_view(), name='register'),
    path('api/storms/<int:storm_id>/', views.storm_details, name='storm_details'),
    path('api/sectors/<int:sector_id>/', views.sector_details, name='sector_details'),
    path('api/specialities/<int:speciality_id>/', views.speciality_details, name='speciality_details'),
    path('api/staff/<int:staff_id>/', views.staff_details, name='staff_details'),
    path('api/conservation_schedules/<int:task_id>/', views.conservation_schedule_details, name='conservation_schedule_details'),
    path('api/parts/<int:part_id>/', views.part_details, name='part_details'),
    path('api/installations/<int:installation_id>/', views.installation_details, name='installation_details'),
    path('api/parts_usages/<int:part_usage_id>/', views.part_usage_details, name='part_usage_details'),
    path('api/damages/<int:damage_id>/', views.damage_details, name='damage_details')
```

## Plik views.py

Plik views.py w aplikacji zawiera logikę obsługi różnych widoków oraz funkcji API. Przede wszystkim, jest odpowiedzialny za renderowanie stron HTML takich jak strona główna, galeria, czy panel administratora. Obejmuje także widoki API, które umożliwiają użytkownikom logowanie, rejestrację, oraz dostęp do szczegółowych informacji o różnych zasobach, takich jak sektory, burze piaskowe, specjalizacje czy pracownicy. Funkcje takie jak home(), admin\_dashboard(), gallery(), oraz information() renderują odpowiednie szablony HTML na podstawie żądań HTTP. W przypadku widoku logowania oraz wylogowania korzysta się z mechanizmów autentykacji Django, aby zarządzać sesjami użytkowników. Z kolei RegisterAPIView pozwala na rejestrację nowych użytkowników. Dodatkowo, dla każdej z tras API np. sector\_details(), storm\_details() widok zwraca dane w formacie JSON. Każdy z tych widoków umożliwia pobranie szczegółów na temat zasobów, takich jak sektory, burze czy pracownicy, w zależności od podanego identyfikatora np. sector\_id, storm\_id. Dzięki dekoratorowi @login\_required, niektóre widoki wymagają wcześniejszego zalogowania się użytkownika, co zapewnia odpowiedni poziom zabezpieczeń i ograniczeń dostępu do danych.

```

from django.shortcuts import render
from rest_framework.views import APIView, View
from rest_framework.response import Response
from rest_framework import status
from services.auth_service import login_user, logout_user, register_user
from services.sector_service import *
from serializers import UserSerializer
from db.models import Sector
from django.http import JsonResponse
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.decorators import login_required
import json
from services.sector_service import *

def home(request):
    return render(request, 'index.html')

def admin_dashboard(request):
    if request.user.is_authenticated and request.user.username == 'admin':
        return render(request, 'admin.html')
    else:
        return render(request, 'index.html')

def gallery(request):
    return render(request, 'gallery.html')

def information(request):
    return render(request, 'information.html')

@method_decorator(csrf_exempt, name='dispatch')
class LoginView(View):
    def post(self, request):
        """Handle user login via AJAX."""
        data = json.loads(request.body.decode("utf-8"))
        username = data.get('username')
        password = data.get('password')

        user = authenticate(request, username=username, password=password)
        if user:
            login(request, user)
            return JsonResponse({"message": "Login successful", "username": user.username})
        return JsonResponse({"error": "Invalid credentials"}, status=401)

@method_decorator(csrf_exempt, name='dispatch')
class LogoutView(View):
    def post(self, request):
        """Handle user logout via AJAX."""
        logout(request)
        return JsonResponse({"message": "Logout successful"})

class RegisterAPIView(APIView):
    def post(self, request):
        """Handle user registration."""
        username = request.data.get('username')
        password = request.data.get('password')
        confirm_password = request.data.get('confirm_password')
        email = request.data.get('email')

        if not username or not password or not confirm_password or not email:
            return Response({"error": "All fields are required."}, status=status.HTTP_400_BAD_REQUEST)

        if password != confirm_password:
            return Response({"error": "Passwords do not match."}, status=status.HTTP_400_BAD_REQUEST)

        try:
            user = register_user(username, password, email)
            serializer = UserSerializer(user)
            return Response(serializer.data, status=status.HTTP_201_CREATED)

```

```

@login_required
def sector_details(request, sector_id):
    try:
        data = get_sector_info(sector_id)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono sektora."}, status=404)
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)

@login_required
def storm_details(request, storm_id):
    try:
        data = get_storm_info(storm_id)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono burzy."}, status=404)
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)

@login_required
def speciality_details(request, speciality_id):
    try:
        data = get_speciality_info(speciality_id)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono specjalności."}, status=404)
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)

@login_required
def staff_details(request, staff_id):
    try:
        data = get_staff_info(staff_id)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono pracownika."}, status=404)
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)

@login_required
def conservation_schedule_details(request, task_id):
    try:
        data = get_conservation_schedule_info(task_id)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono naprawy."}, status=404)
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)

@login_required
def part_details(request, part_id):
    try:
        data = get_part_info(part_id)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono części."}, status=404)
    except Exception as e:

```



```

@login_required
def part_details(request, part_id):
    try:
        data = get_part_info(part_id)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono części."}, status=404)
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)

@login_required
def installation_details(request, installation_id):
    try:
        data = get_installation_info(installation_id)
        print(data)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono instalacji."}, status=404)
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)

@login_required
def part_usage_details(request, part_usage_id):
    try:
        data = get_partsusage_info(part_usage_id)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono użycia części."}, status=404)
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)

@login_required
def damage_details(request, damage_id):
    try:
        data = get_damage_info(damage_id)
        if data:
            return JsonResponse(data) # Zwracanie szczegółów w formacie JSON
        else:
            return JsonResponse({"error": "Nie znaleziono uszkodzenia."}, status=404)
    except Exception as e:
        return JsonResponse({"error": str(e)}, status=500)

```

## Plik admin.py

Plik `admin.py` służy do rejestrowania modeli w panelu administracyjnym, co umożliwia zarządzanie danymi tych modeli za pośrednictwem interfejsu administracyjnego Django. W tym przypadku, plik ten rejestruje kilka modeli, takich jak `Storm`, `Sector`, `Speciality`, `Staff`, `ConservationSchedule`, `Part`, `Installation`, `PartsUsage`, oraz `Damage`, które są związane z projektem Mars-Dust. Każdy z tych modeli jest importowany z modułu `db.models` i następnie rejestrowany w panelu administracyjnym za pomocą funkcji `admin.site.register()`. Dzięki temu, administratorzy systemu mogą dodawać, edytować i usuwać rekordy w tabelach odpowiadających tym modelom bez konieczności pisania zapytań SQL. Rejestracja modeli w Django Admin zapewnia prosty i wygodny sposób interakcji z bazą danych, pozwalając na szybkie zarządzanie danymi oraz kontrolę nad nimi. Jest to szczególnie przydatne w systemach, które wymagają łatwego dostępu do danych przez użytkowników administracyjnych. W przypadku zakomentowanych linii z `os.environ.setdefault()` oraz `django.setup()`, te linie mogłyby być używane do ustawienia konfiguracji Django w kontekście skryptów zewnętrznych (np. uruchamiania aplikacji Django w środowiskach, które nie są bezpośrednio powiązane z serwerem). Jednak w przypadku zwykłego pliku `admin.py` takie linie są zbędne i nie wpływają na jego działanie w typowym środowisku Django.

```

from django.contrib import admin
from db.models import Sector, Storm, Speciality, Staff, ConservationSchedule, Part, Installation, PartsUsage, Damage

#import os
#os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'marsdust.settings')

#import django
#django.setup()

admin.site.register(Storm)
admin.site.register(Sector)
admin.site.register(Speciality)
admin.site.register(Staff)
admin.site.register(ConservationSchedule)
admin.site.register(Part)
admin.site.register(Installation)
admin.site.register(PartsUsage)
admin.site.register(Damage)

```

## Plik serializers.py

Plik `serializers.py` w Django REST Framework pełni kluczową rolę w konwersji obiektów bazodanowych na format JSON oraz odwrotnie, co umożliwia efektywną wymianę danych między backendem a frontendem lub innymi aplikacjami korzystającymi z API. W aplikacji Mars-Dust definiuje on serializatory dla różnych modeli, które strukturalizują dane i ułatwiają ich obsługę w zapytaniach API. Podstawowe serializatory, takie jak `StormSerializer`, `SpecialitySerializer` i `SectorSerializer`, konwertują wszystkie pola danych bez dodatkowych zależności. Bardziej złożone struktury, takie jak `StaffSerializer` i `ConservationScheduleSerializer`, wykorzystują zagnieżdżone serializatory, umożliwiając zwracanie danych powiązanych obiektów, na przykład pracowników wraz z ich specjalizacjami oraz harmonogramów konserwacji ze szczegółowymi informacjami o przypisanym personelu. `PartSerializer` oraz `InstallationSerializer` wykorzystują mechanizm `StringRelatedField` do reprezentacji powiązanych obiektów w postaci tekstowej, co minimalizuje nadmierne zagnieżdżanie i poprawia czytelność danych. W przypadku `PartsUsageSerializer` oraz `DamageSerializer` zastosowano pełne zagnieżdżone serializatory, co pozwala na jednoczesne zwrócenie informacji o zużytych częściach, ich powiązanych instalacjach, a także szczegółach uszkodzeń i ewentualnych napraw. `UserSerializer` natomiast odpowiada za konwersję danych użytkownika, ograniczając dostęp do podstawowych informacji, takich jak nazwa użytkownika oraz adres e-mail. Struktura tego pliku pozwala na efektywne zarządzanie danymi oraz optymalizuje procesy komunikacji pomiędzy aplikacją a interfejsem użytkownika. Dzięki zastosowaniu Django REST Framework API zwraca dane w uporządkowany sposób, co znacząco ułatwia implementację zarówno po stronie klienta, jak i serwera.

```

from rest_framework import serializers
from db.models import (
    Storm, Speciality, Staff, ConservationSchedule,
    Part, Installation, Sector, PartsUsage, Damage
)
from django.contrib.auth.models import User

class StormSerializer(serializers.ModelSerializer):
    class Meta:
        model = Storm
        fields = '__all__'

class SpecialitySerializer(serializers.ModelSerializer):
    class Meta:
        model = Speciality
        fields = '__all__'

class StaffSerializer(serializers.ModelSerializer):
    speciality = SpecialitySerializer()

    class Meta:
        model = Staff
        fields = '__all__'

class ConservationScheduleSerializer(serializers.ModelSerializer):
    staff = StaffSerializer()

    class Meta:
        model = ConservationSchedule
        fields = '__all__'

class PartSerializer(serializers.ModelSerializer):
    installation = serializers.StringRelatedField()

    class Meta:
        model = Part
        fields = '__all__'

class InstallationSerializer(serializers.ModelSerializer):
    sector = serializers.StringRelatedField()

    class Meta:
        model = Installation
        fields = '__all__'

class SectorSerializer(serializers.ModelSerializer):
    class Meta:
        model = Sector
        fields = '__all__'

class PartsUsageSerializer(serializers.ModelSerializer):
    installation = InstallationSerializer()
    part = PartSerializer()

    class Meta:
        model = PartsUsage
        fields = '__all__'

class DamageSerializer(serializers.ModelSerializer):
    part = PartSerializer()
    queued_task = ConservationScheduleSerializer()
    cause = StormSerializer()

    class Meta:
        model = Damage
        fields = '__all__'

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['username', 'email']

```

### **Plik 0001\_initial.py**

Plik 0001\_initial.py znajdujący się w folderze db jest pierwszą migracją bazy danych w aplikacji Mars-Dust. Został wygenerowany przez Django i zawiera instrukcje do utworzenia tabel w bazie danych oraz określenia ich struktury i relacji. Migracje pozwalają na wersjonowanie schematu bazy danych, co ułatwia jego modyfikację i rozwój aplikacji. W pliku tym definiowane są modele reprezentujące kluczowe elementy systemu. Model Installation przechowuje informacje o instalacjach, a każdy obiekt tej klasy zawiera identyfikator oraz nazwę. Model Sector opisuje sektory w aplikacji, zawierając dane o nazwie, opisie, współrzędnych geograficznych oraz granicach sektorów. Model Speciality reprezentuje specjalizacje pracowników, a Storm przechowuje szczegółowe informacje o burzach piaskowych, takie jak rok marsjański, długość trwania, lokalizacja, moc oraz ewentualne szkody wyrządzone przez kamienie. Model Part odnosi się do części składowych instalacji, a PartsUsage śledzi ich użycie, wskazując zarówno instalację, jak i część. Instalacje są bezpośrednio powiązane z sektorami, co umożliwia organizację infrastruktury. Model Staff zawiera informacje o pracownikach, ich imionach, nazwiskach, cechach oraz przypisanych specjalizacjach. ConservationSchedule przechowuje harmonogramy konserwacji, powiązując zadania z odpowiedzialnym personelem i określając czas ich realizacji. Model Damage jest kluczowy dla monitorowania uszkodzeń i obejmuje informacje o przypuszczalnym lub zgłoszonym charakterze uszkodzenia, jego poziomie powagi oraz powiązaniu z określoną częścią. Może także wskazywać zadanie konserwacyjne zaplanowane w związku z daną usterką oraz burzę piaskową, która mogła być jej przyczyną. Plik ten określa także relacje między modelami. Większość powiązań opiera się na kluczu obcym, co oznacza, że jeden obiekt może być powiązany z innym za pomocą identyfikatora. Niektóre relacje, jak w przypadku PartsUsage czy Part, wykorzystują OneToOneField, co oznacza, że każda część jest unikalnie przypisana do jednej instalacji. Niektóre pola mogą być puste lub mieć wartość null, np. queued\_task w Damage, co oznacza, że uszkodzenie nie zawsze musi być związane z planowaną naprawą. Plik 0001\_initial.py jest niezbędny do poprawnego działania bazy danych i stanowi podstawę dla kolejnych migracji w systemie, umożliwiając skalowanie i rozwój aplikacji bez konieczności ręcznej modyfikacji struktury bazy danych.

### **Plik 0002\_sector\_parent.py**

Plik 0002\_sector\_parent.py to kolejna migracja w projekcie Mars-Dust, która rozszerza model Sector o nową funkcjonalność. Został on wygenerowany przez Django i stanowi uzupełnienie wcześniejszej migracji 0001\_initial.py. Główna zmiana wprowadzona przez tę migrację to dodanie pola parent w modelu Sector, które tworzy relację hierarchiczną między sektorami. Nowe pole jest kluczem obcym wskazującym na inny obiekt Sector, co oznacza, że każdy sektor może mieć nadrzędny sektor, tworząc strukturę drzewiastą. Pola blank=True i null=True oznaczają, że wartość parent nie jest obowiązkowa – sektor może istnieć samodzielnie lub należeć do większej jednostki organizacyjnej. Parametr on\_delete=models.CASCADE powoduje, że usunięcie sektora nadrzędnego automatycznie usuwa wszystkie sektory podrzędne. Dodatkowo related\_name='sub\_sectors' umożliwia łatwe uzyskanie dostępu do sektorów podrzędnych poprzez odniesienie się do nich z poziomu sektora nadrzędnego.

### **Plik 0003\_alter\_damage\_part\_alter\_installation\_sector\_and\_more.py**

Plik 0003\_alter\_damage\_part\_alter\_installation\_sector\_and\_more.py to kolejna migracja w projekcie Mars-Dust, która wprowadza modyfikacje w relacjach między kluczowymi modelami bazy danych. Został on wygenerowany przez Django i stanowi kontynuację zmian wprowadzonych w migracji 0002\_sector\_parent.py. Główna zmiana w tej migracji polega na dostosowaniu pól powiązanych w modelach Damage, Installation, Part oraz PartsUsage. W modelu Damage pole part zostało określone jako relacja jeden do jednego z modelem Part, co zapewnia unikalne powiązanie uszkodzeń z konkretną częścią i ułatwia dostęp do powiązanych informacji poprzez nazwę Damage. Podobne zmiany wprowadzono w modelu Installation, gdzie sektor został określony jako relacja jeden do jednego, co jednoznacznie przypisuje każdą instalację do konkretnego sektora i pozwala na łatwy dostęp do powiązanych danych za pomocą oznaczenia Installation. Model Part został powiązany z instalacją w taki sam sposób, co zapewnia jednoznaczną relację między tymi elementami infrastruktury. Zmiany objęły również model PartsUsage, gdzie zarówno instalacja, jak i część, zostały zdefiniowane jako relacje jeden do jednego, co umożliwia precyzyjne śledzenie wykorzystania części w ramach instalacji. Wprowadzone modyfikacje zwiększają spójność danych, upraszczają zapytania do bazy oraz poprawiają organizację struktury danych w systemie.

### **Plik models.py**

Plik models.py w projekcie Mars-Dust definiuje strukturę bazy danych, opisując kluczowe modele i ich relacje. Model Storm przechowuje informacje o burzach pyłowych na Marsie, zawierając takie pola jak identyfikator burzy, rok marsjański, współrzędne centroidu, czas trwania oraz moc burzy. Model Speciality definiuje specjalizacje członków personelu, które są następnie powiązane z modelem Staff, przechowującym dane o pracownikach, takie jak imię, nazwisko, specjalizacja oraz cechy charakterystyczne. Model ConservationSchedule opisuje harmonogramy konserwacji, łącząc zadania z personelem oraz określając czas ich rozpoczęcia i zakończenia. Model Installation reprezentuje instalacje na Marsie i jest powiązany jeden do jednego z modelem Sector, który zawiera szczegółowe informacje o sektorach, ich położeniu geograficznym oraz możliwość hierarchicznego przypisania do nadrzędnych jednostek. Model Part przechowuje dane o częściach składających się na instalacje, a PartsUsage rejestruje ich wykorzystanie w ramach infrastruktury. Model Damage odpowiada za ewidencję uszkodzeń części, określając ich przyczynę, powiązane zadania konserwacyjne oraz poziom dotkliwości. Wprowadzone relacje między modelami zapewniają integralność danych oraz umożliwiają efektywne zarządzanie infrastrukturą i monitorowanie warunków na Marsie.

```

from django.db import models

class Storm(models.Model):
    storm_id = models.AutoField(primary_key=True) # Auto-Increment ID
    mars_year = models.IntegerField() # Integer, NOT NULL
    sol = models.CharField(max_length=3)
    member_id = models.IntegerField() # String with max length of 3
    centroid_latitude = models.DecimalField(max_digits=14, decimal_places=6)
    centroid_longitude = models.DecimalField(max_digits=14, decimal_places=6)
    duration = models.DecimalField(max_digits=14, decimal_places=6)
    mission_subphase = models.CharField(max_length=255)
    spread_latitude = models.DecimalField(max_digits=14, decimal_places=6)
    spread_longitude = models.DecimalField(max_digits=14, decimal_places=6)
    power = models.IntegerField()
    stone_damage = models.BooleanField()

    def __str__(self):
        return str(f"Storm {self.storm_id} - Coordinates {self.centroid_latitude} x {self.centroid_longitude}")

class Speciality(models.Model):
    speciality_id = models.AutoField(primary_key=True) # Auto-Increment field
    name = models.CharField(max_length=255) # Name, NOT NULL

    def __str__(self):
        return str(f"{self.name}")

class Staff(models.Model):
    staff_id = models.AutoField(primary_key=True) # Auto-Increment field
    name = models.CharField(max_length=255) # Staff name, NOT NULL
    surname = models.CharField(max_length=255) # Surname, NOT NULL
    speciality = models.ForeignKey('Speciality', on_delete=models.CASCADE) # ForeignKey to Specialities
    traits = models.CharField(max_length=255) # Traits, NOT NULL

    def __str__(self):
        return str(f"{self.name} {self.surname}")

class ConservationSchedule(models.Model):
    task_id = models.AutoField(primary_key=True) # Auto-Increment field
    staff = models.ForeignKey('Staff', on_delete=models.CASCADE) # ForeignKey to Staff
    start_time = models.DateTimeField() # Start timestamp, NOT NULL
    end_time = models.DateTimeField() # End timestamp, NOT NULL

    def __str__(self):
        return str(f"Task {self.task_id} - Staff {self.staff.name}")

class Part(models.Model):
    part_id = models.AutoField(primary_key=True) # Primary key
    installation = models.OneToOneField('Installation', on_delete=models.CASCADE, related_name='Part')
    name = models.CharField(max_length=255)

    def __str__(self):
        return str(f"{self.name}")

class Installation(models.Model):
    installation_id = models.AutoField(primary_key=True) # Placeholder for "Installation_Obj"
    sector = models.OneToOneField('Sector', on_delete=models.CASCADE, related_name='Installation')
    name = models.CharField(max_length=255)

    def __str__(self):
        return str(f"{self.name}")

```

```

class Sector(models.Model):
    sector_id = models.AutoField(primary_key=True)
    name = models.CharField(max_length=255)
    description = models.CharField(max_length=1023)
    max_latitude = models.IntegerField()
    min_latitude = models.IntegerField()
    max_longitude = models.IntegerField()
    min_longitude = models.IntegerField()
    parent = models.ForeignKey('self', on_delete=models.CASCADE, related_name='sub_sectors', null=True, blank=True)

    def __str__(self):
        return str(f"{self.sector_id} - {self.name}")

class PartsUsage(models.Model):
    part_usage_id = models.AutoField(primary_key=True)
    installation = models.OneToOneField('Installation', on_delete=models.CASCADE, related_name='PartsUsage') # Foreign key to Installation
    part = models.OneToOneField('Part', on_delete=models.CASCADE, related_name='PartsUsage') # Foreign key to Part

    def __str__(self):
        return str(f"Usage {self.part_usage_id}: Installation {self.installation} - Part {self.part}")

class Damage(models.Model):
    damage_id = models.AutoField(primary_key=True)
    part = models.OneToOneField('Part', on_delete=models.CASCADE, related_name='Damage') # Foreign key to PartsInt
    presumed_or_reported = models.BooleanField() # Converted from NUMBER(1)
    queued_task = models.ForeignKey('ConservationSchedule', null=True, blank=True, on_delete=models.SET_NULL)
    cause = models.ForeignKey('Storm', null=True, blank=True, on_delete=models.SET_NULL)
    severity = models.IntegerField() # Severity level, NOT NULL

    def __str__(self):
        return str(f"Damaged Part {self.part}")

```

## Plik tablesSetup.sql

Plik tablesSetup.sql w projekcie Mars-Dust odpowiada za tworzenie tabel w bazie danych, definiowanie typów obiektów oraz określenie relacji między nimi. W pierwszej kolejności tworzona jest tabela Storms, która przechowuje informacje o burzach, takie jak identyfikator burzy, rok na Marsie, współrzędne geograficzne, obszar, intensywność oraz inne parametry związane z burzą, przy czym Storm\_ID pełni rolę klucza głównego, zapewniając unikalność rekordów. Kolejnymi elementami są zdefiniowane typy obiektów InstallationType\_Obj i Sector\_Obj, które stanowią definicje dla typów instalacji oraz sektorów, wykorzystywane później do przechowywania tych danych w tabelach. Tabela Installations przechowuje dane o instalacjach, powiązanych z określonymi typami oraz sektorami, a także wykorzystuje zagnieżdżoną tabelę Sector\_Table\_varname, pozwalając na przechowywanie wielu sektorów w ramach jednej instalacji. Kolejne tabele, takie jak PartsInternalCodes, PartExternalCodes, Specialities, Staff, PartsUsage, ConservationSchedule oraz DamagedParts, zawierają dane dotyczące części, specjalności pracowników, harmonogramów konserwacji oraz uszkodzeń części. Tabele te są ze sobą powiązane za pomocą kluczy obcych, co zapewnia spójność danych w całym systemie. Plik definiuje również złożone relacje, zarówno typu jeden do wielu, jak i one-to-one, umożliwiając efektywne zarządzanie danymi. Ponadto, w pliku znajdują się zakomentowane komendy DROP, które pozwalają na usunięcie tabel w razie potrzeby ich ponownego utworzenia.

```
--DROP TABLE PartsUsage CASCADE CONSTRAINTS;
--DROP TABLE ConservationSchedule CASCADE CONSTRAINTS;
--DROP TABLE DamagedParts CASCADE CONSTRAINTS;
--DROP TABLE Specialities CASCADE CONSTRAINTS;
--DROP TABLE Staff CASCADE CONSTRAINTS;
--DROP TABLE PartsInternalCodes CASCADE CONSTRAINTS;
--DROP TABLE PartExternalCodes CASCADE CONSTRAINTS;
--DROP TABLE Installations CASCADE CONSTRAINTS;
--DROP TABLE InstallationTypes CASCADE CONSTRAINTS;
--DROP TABLE Sectors CASCADE CONSTRAINTS;
--DROP TABLE Storms CASCADE CONSTRAINTS;

CREATE TABLE Storms (
    Storm_ID NUMBER(11) DEFAULT seq_Storms.NEXTVAL,
    Mars_Year NUMBER(10) NOT NULL,
    Sol VARCHAR2(3) NOT NULL,
    Mission_subphase VARCHAR2(255) NOT NULL,
    Solar_longitude_Is NUMBER(14,6) NOT NULL,
    Centroid_longitude NUMBER(14,6) NOT NULL,
    Centroid_latitude NUMBER(14,6) NOT NULL,
    Area NUMBER(14,6) NOT NULL,
    Member_ID VARCHAR2(255) NOT NULL,
    Sequence_ID VARCHAR2(20),
    Max_latitude NUMBER(14,6) NOT NULL,
    Min_latitude NUMBER(14,6) NOT NULL,
    Confidence_interval NUMBER(10) NOT NULL,
    Missing_data NUMBER(1) NOT NULL,
    CONSTRAINT pk_Storms PRIMARY KEY (Storm_ID)
);

CREATE OR REPLACE TYPE InstallationType_Obj AS OBJECT (
    Type_ID NUMBER(10),
    Name VARCHAR2(255)
);
/

CREATE OR REPLACE TYPE Sector_Obj AS OBJECT (
    Sector_ID NUMBER(10),
    MaxLatitude NUMBER(10),
    MinLatitude NUMBER(10),
    MaxLongitude NUMBER(10),
    MinLongitude NUMBER(10)
);
/

CREATE OR REPLACE TYPE InstallationType_VARRAY AS VARRAY(5) OF (InstallationType_Obj);
/
CREATE OR REPLACE TYPE Sector_Table AS TABLE OF Sector_Obj;
/

CREATE OR REPLACE TYPE Installation_Obj AS OBJECT (
    Installation_ID NUMBER(10),
    Type InstallationType_Obj,
    Name VARCHAR2(255)
);
/
CREATE OR REPLACE TYPE installation_table AS TABLE OF installation_obj;
/

CREATE TABLE Installations (
    Installation Installation_Obj,
    Sector_Table_varname Sector_Table
) NESTED TABLE Sector_Table_varname STORE AS Sector_NT_Store;
/
```

```

);
CONSTRAINT pk_PartsInternalCodes PRIMARY KEY (Part_ID, Internal_ID)
);

CREATE TABLE PartExternalCodes (
    PartID NUMBER(10) DEFAULT seq_PartExternalCodes.NEXTVAL,
    Name VARCHAR2(255) NOT NULL,
    CONSTRAINT pk_PartExternalCodes PRIMARY KEY (PartID)
);

CREATE TABLE Specialities (
    Speciality_ID NUMBER(10) DEFAULT seq_Specialities.NEXTVAL,
    Name VARCHAR2(255) NOT NULL,
    CONSTRAINT pk_Specialities PRIMARY KEY (Speciality_ID)
);

CREATE TABLE Staff (
    Staff_ID NUMBER(11) DEFAULT seq_Staff.NEXTVAL,
    Name VARCHAR2(255) NOT NULL,
    Surname VARCHAR2(255) NOT NULL,
    Speciality_ID NUMBER(10) NOT NULL,
    Traits VARCHAR2(255) NOT NULL,
    CONSTRAINT pk_Staff PRIMARY KEY (Staff_ID),
    CONSTRAINT fk_Staff_Specialities FOREIGN KEY (Speciality_ID) REFERENCES Specialities(Speciality_ID)
);

CREATE TABLE PartsUsage (
    Installation_ID NUMBER(10) DEFAULT seq_PartsUsage.NEXTVAL,
    Part_ID NUMBER(10) DEFAULT seq_PartsUsage.NEXTVAL,
    Internal_ID NUMBER(10) DEFAULT seq_PartsUsage.NEXTVAL,
    CONSTRAINT pk_PartsUsage PRIMARY KEY (Installation_ID, Part_ID, Internal_ID),
    --CONSTRAINT fk_PartsUsage_Installations FOREIGN KEY (Installation_ID) REFERENCES Installations(Installation_ID)
    CONSTRAINT fk_PartsUsage_PartsInternalCodes FOREIGN KEY (Part_ID, Internal_ID) REFERENCES PartsInternalCodes(Part_ID, Internal_ID)
);

CREATE TABLE ConservationSchedule (
    Task_ID NUMBER(10) DEFAULT seq_ConservationSchedule.NEXTVAL,
    Staff_ID NUMBER(10) NOT NULL,
    StartTime TIMESTAMP NOT NULL,
    EndTime TIMESTAMP NOT NULL,
    CONSTRAINT pk_ConservationSchedule PRIMARY KEY (Task_ID),
    CONSTRAINT fk_ConservationSchedule_Staff FOREIGN KEY (Staff_ID) REFERENCES Staff(Staff_ID)
);

CREATE TABLE DamagedParts (
    Part_ID NUMBER(10) DEFAULT seq_DamagedParts.NEXTVAL,
    Internal_ID NUMBER(10) DEFAULT seq_DamagedParts.NEXTVAL,
    PresumedOrReported NUMBER(1) NOT NULL,
    QueuedTask NUMBER(10),
    Cause_ID VARCHAR(7),
    Severity NUMBER(10) NOT NULL,
    CONSTRAINT pk_DamagedParts PRIMARY KEY (Part_ID, Internal_ID),
    CONSTRAINT fk_DamagedParts_PartsInternalCodes FOREIGN KEY (Part_ID, Internal_ID) REFERENCES PartsInternalCodes(Part_ID, Internal_ID),
    CONSTRAINT fk_DamagedParts_ConservationSchedule FOREIGN KEY (QueuedTask) REFERENCES ConservationSchedule(Task_ID),
    --CONSTRAINT fk_DamagedParts_Storms FOREIGN KEY (Cause_ID) REFERENCES Storms(Sequence_ID)
    --CONSTRAINT fk_DamagedParts_Staff FOREIGN KEY (Cause_ID) REFERENCES Staff(Staff_ID)
) ORGANIZATION INDEX;

```

## Plik auth\_service.py

Plik `auth_service.py` w projekcie `Mars-Dust` odpowiada za zarządzanie procesami logowania, rejestracji oraz wylogowywania użytkowników. Wykorzystuje on wbudowane funkcje Django do autentykacji oraz obsługi sesji użytkownika. W pierwszej funkcji `login_user`, użytkownik jest autentykowany za pomocą dostarczonych danych, czyli nazwy użytkownika oraz hasła. Jeśli dane są poprawne, użytkownik jest logowany, a funkcja zwraca obiekt użytkownika. W przeciwnym razie zwraca wartość `None`, co oznacza, że logowanie się nie powiodło. Funkcja `logout_user` odpowiedzialna jest za wylogowanie użytkownika z aplikacji, co realizowane jest poprzez wywołanie metody `logout` Django. Ostatnia funkcja `register_user` umożliwia rejestrację nowego użytkownika. Sprawdza ona, czy wszystkie wymagane dane (nazwa użytkownika, hasło, email) zostały dostarczone, a następnie weryfikuje, czy dany użytkownik lub email już istnieją w bazie. Jeśli tak, funkcja zgłasza wyjątek z odpowiednim komunikatem. Jeśli dane są prawidłowe, tworzy nowego użytkownika za pomocą metody `create_user` i zwraca obiekt użytkownika. Funkcje te są pomocnicze w zapewnianiu bezpieczeństwa i integralności procesów logowania oraz rejestracji w aplikacji, pozwalając na łatwe zarządzanie sesjami użytkowników.



```

from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.models import User

def login_user(request, username, password):
    """Authenticate and log in a user."""
    user = authenticate(request, username=username, password=password)
    if user:
        login(request, user)
        return user
    return None

def logout_user(request):
    """Log out a user."""
    logout(request)

def register_user(username, password, email=None):
    """Register a new user."""
    if not username or not password or not email:
        raise ValueError("Proszę podać wszystkie dane: nazwa użytkownika, email, hasło.")

    if User.objects.filter(username=username).exists():
        raise ValueError("Nazwa użytkownika jest już zajęta.")

    if User.objects.filter(email=email).exists():
        raise ValueError("Adres email jest już zajęty.")

    user = User.objects.create_user(username=username, password=password, email=email)
    return user

```

## Plik sector\_service.py

Plik sector\_service.py zawiera funkcje do pobierania informacji o różnych obiektach w systemie, takich jak burze, specjalizacje, personel, harmonogramy konserwacji, części, instalacje, sektory i ich uszkodzenia. Każda funkcja realizuje zapytanie do bazy danych za pomocą Django ORM, aby znaleźć odpowiedni obiekt na podstawie dostarczonego identyfikatora ID, a następnie zwraca szczegółowe dane o tym obiekcie. Główne funkcje:

1. `get_storm_info(id)` Zwraca informacje o burzy na podstawie jej ID, w tym dane takie jak współrzędne, rok marsjański, długość trwania burzy oraz ewentualne uszkodzenia, jeśli występują.
2. `get_speciality_info(id)` Pobiera informacje o specjalizacji, w tym nazwę specjalizacji oraz przypisanego personelu, jeśli istnieje.
3. `get_staff_info(id)` Zwraca dane o pracowniku, takie jak imię, nazwisko, przypisaną specjalizację oraz zaplanowane zadania konserwacyjne.
4. `get_conservationchedule_info(id)` Zawiera szczegóły dotyczące zadania konserwacyjnego, w tym czas rozpoczęcia i zakończenia, przypisanego pracownika oraz ewentualne uszkodzenia.
5. `get_part_info(id)` Pobiera dane o części, w tym nazwę części, powiązaną instalację, użycie części oraz ewentualne uszkodzenia.
6. `get_installation_info(id)` Zwraca dane o instalacji, takie jak przypisany sektor, użycie części i inne powiązane informacje.
7. `get_sector_info(id)` Zawiera szczegóły dotyczące sektora, w tym jego opis, współrzędne i powiązaną instalację, jeśli istnieje.
8. `get_partsusage_info(id)` Pobiera informacje o użyciu części w instalacji, w tym powiązaną część oraz instalację.
9. `get_damage_info(id)` Zwraca szczegóły dotyczące uszkodzenia, w tym przyczynę, jak burzę, powiązaną część i zadania konserwacyjne.

W każdej funkcji najpierw sprawdzane jest, czy obiekt o podanym ID istnieje w bazie danych. Jeśli nie, zwracany jest błąd z informacją o braku obiektu. Jeśli obiekt istnieje, funkcja zwraca szczegółowe informacje o obiekcie w postaci słownika.

```

from db.models import Storm, Speciality, Staff, ConservationSchedule, Part, Installation, Sector, PartUsage, Damage

def get_storm_info(id):
    try:
        storm = Storm.objects.get(storm_id=id)
        damage = Damage.objects.filter(cause=storm.storm_id).first()

        return {
            "Info": storm.__str__(),
            "Centroid latitude": storm.centroid_latitude,
            "Centroid longitude": storm.centroid_longitude,
            "Duration": storm.duration,
            "Mars year": storm.mars_year,
            "Number ID": storm.member_id,
            "Mission subphase": storm.mission_subphase,
            "Power": storm.power,
            "Soil": storm.soil,
            "Spread latitude": storm.spread_latitude,
            "Spread longitude": storm.spread_longitude,
            "Stone damage": storm.stone_damage,
            "Damage": damage.__str__() if damage else "Brak uszkodzen",
            "Damage ID": damage.damage_id if damage else "Brak uszkodzen"
        }
    except Storm.DoesNotExist:
        return {"error": "Burza nie istnieje"}

def get_speciality_info(id):
    try:
        speciality = Speciality.objects.get(speciality_id=id)
        staff = Staff.objects.filter(speciality=speciality.speciality_id).first()

        return {
            "Info": speciality.__str__(),
            "Name": speciality.name,
            "Staff": staff.__str__() if staff else "Brak personelu",
            "Staff ID": staff.staff_id if staff else "Brak personelu"
        }
    except Speciality.DoesNotExist:
        return {"error": "Specjalizacja nie istnieje"}

def get_staff_info(id):
    try:
        staff = Staff.objects.get(staff_id=id)
        conservation_schedule = ConservationSchedule.objects.filter(staff=staff.staff_id).first()

        return {
            "Info": staff.__str__(),
            "Speciality": staff.speciality.__str__() if staff.speciality else "Brak specjalizacji",
            "Speciality ID": staff.speciality.speciality_id if staff.speciality else "Brak specjalizacji",
            "Name": staff.name,
            "Surname": staff.surname,
            "Traits": staff.traits,
            "Conservation schedule": conservation_schedule.__str__() if conservation_schedule else "Brak napraw",
            "Conservation schedule ID": conservation_schedule.task_id if conservation_schedule else "Brak napraw"
        }
    except Staff.DoesNotExist:
        return {"error": "Osoba nie istnieje"}

def get_conservation_schedule_info(id):
    try:
        conservation_schedule = ConservationSchedule.objects.get(task_id=id)
        damage = Damage.objects.filter(queued_task=conservation_schedule.task_id).first()

        return {
            "Info": conservation_schedule.__str__(),
            "Staff": conservation_schedule.staff.__str__() if conservation_schedule.staff else "Brak personelu",
            "Staff ID": conservation_schedule.staff.staff_id if conservation_schedule.staff else "Brak personelu",
        }
    except ConservationSchedule.DoesNotExist:
        return {"error": "Naprawa nie istnieje"}

```

```

def get_conservation_schedule_info(id):
    try:
        conservation_schedule = ConservationSchedule.objects.get(task_id=id)
        damage = Damage.objects.filter(queued_task=conservation_schedule.task_id).first()

        return {
            "Info": conservation_schedule.__str__(),
            "Staff": conservation_schedule.staff.__str__() if conservation_schedule.staff else "Brak personelu",
            "Staff ID": conservation_schedule.staff.staff_id if conservation_schedule.staff else "Brak personelu",
            "Start time": conservation_schedule.start_time,
            "End time": conservation_schedule.end_time,
            "Damage": damage.__str__() if damage else "Brak uszkodzen",
            "Damage ID": damage.damage_id if damage else "Brak uszkodzen"
        }
    except ConservationSchedule.DoesNotExist:
        return {"error": "Naprawa nie istnieje"}

def get_part_info(id):
    try:
        part = Part.objects.get(part_id=id)
        part_usage = PartUsage.objects.filter(part=part).first()
        damage = Damage.objects.filter(part = part).first()

        return {
            "Info": part.__str__(),
            "Installation": part.installation.__str__() if part.installation else "Brak instalacji",
            "Installation ID": part.installation.installation_id if part.installation else "Brak instalacji",
            "Name": part.name,
            "Part usage": part_usage.__str__() if part_usage else "Brak uzycia czesci",
            "Part usage ID": part_usage.part_usage_id if part_usage else "Brak uzycia czesci",
            "Damage": damage.__str__() if damage else "Brak napraw",
            "Damage ID": damage.damage_id if damage else "Brak napraw"
        }
    except Part.DoesNotExist:
        return {"error": "Czesc nie istnieje"}

def get_installation_info(id):
    try:
        installation = Installation.objects.get(installation_id=id)
        part_usage = PartUsage.objects.filter(installation=installation.installation_id).first()

        return {
            "Info": installation.__str__(),
            "Sector": installation.sector.__str__() if installation.sector else "Brak sektora",
            "Sector ID": installation.sector.sector_id if installation.sector else "Brak sektora",
            "Name": installation.name,
            "Part usage": part_usage.__str__() if part_usage else "Brak uzycia czesci",
            "Part usage ID": part_usage.part_usage_id if part_usage else "Brak uzycia czesci",
        }
    except Installation.DoesNotExist:
        return {"error": "Instalacja nie istnieje"}

def get_sector_info(id):
    try:
        sector = Sector.objects.get(sector_id=id)
        installation = Installation.objects.filter(sector=sector).first()

        return {
            "Info": sector.__str__(),
            "Sector name": sector.name,
            "Description": sector.description,
            "Min latitude": sector.min_latitude,
            "Max latitude": sector.max_latitude,
            "Min longitude": sector.min_longitude,
            "Max longitude": sector.max_longitude,
            "Installation": installation.__str__() if installation else "Brak instalacji",
            "Installation ID": installation.installation_id if installation else "Brak instalacji"
        }
    except Sector.DoesNotExist:
        return {"error": "Sektor nie istnieje"}

def get_partsusage_info(id):
    try:
        part_usage = PartUsage.objects.get(part_usage_id=id)
    except PartUsage.DoesNotExist:
        return {"error": "Uzycie czesci nie istnieje"}

```

```

def get_partsusage_info(id):
    try:
        part_usage = PartsUsage.objects.get(part_usage_id=id)

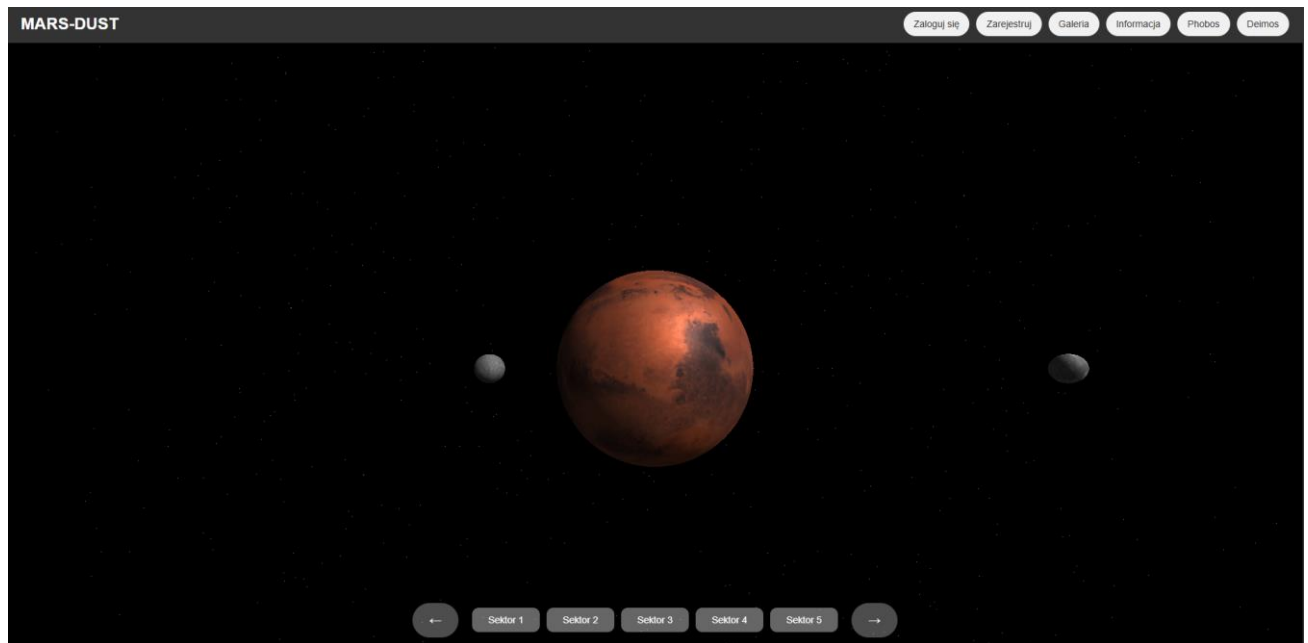
        return {
            "Info": part_usage.__str__(),
            "Part": part_usage.part.__str__() if part_usage.part else "Brak części",
            "Part ID": part_usage.part.part_id if part_usage.part else "Brak części",
            "Installation": part_usage.installation.__str__() if part_usage.installation else "Brak instalacji",
            "Installation ID": part_usage.installation.installation_id if part_usage.installation else "Brak instalacji"
        }
    except PartsUsage.DoesNotExist:
        return {"error": "Użycie części nie istnieje"}

def get_damage_info(id):
    try:
        damage = Damage.objects.get(damage_id=id)

        return {
            "Info": damage.__str__(),
            "Cause": damage.cause.__str__() if damage.cause else "Brak burzy",
            "Cause ID": damage.cause.storm_id if damage.cause else "Brak burzy",
            "Part": damage.part.__str__() if damage.part else "Brak części",
            "Part ID": damage.part.part_id if damage.part else "Brak części",
            "Queued task": damage.queued_task.__str__() if damage.queued_task else "Brak napraw",
            "Queued task ID": damage.queued_task.task_id if damage.queued_task else "Brak napraw",
            "Presumed or reported": damage.presumed_or_reported,
            "Severity": damage.severity
        }
    except Damage.DoesNotExist:
        return {"error": "Uszkodzenie nie istnieje"}

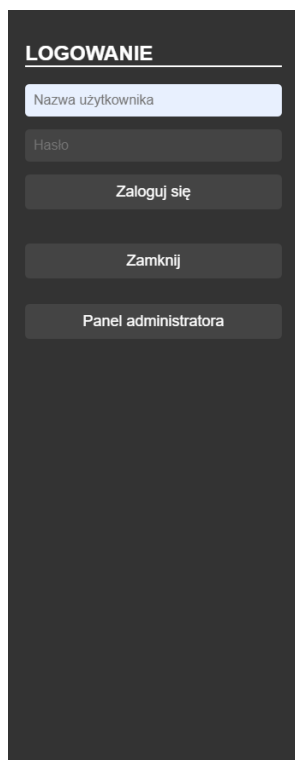
```

# Aplikacja



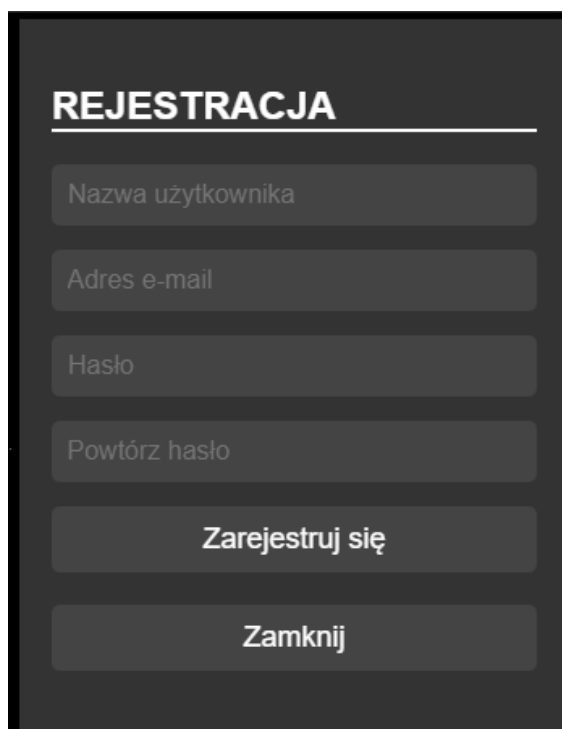
Aplikacja Mars-Dust charakteryzuje się nowoczesnym i intuicyjnym interfejsem użytkownika, zaprojektowanym z myślą o łatwej nawigacji i atrakcyjnej wizualizacji danych. Centralnym elementem aplikacji jest interaktywny model 3D planety Mars, który można obracać za pomocą myszy oraz powiększać i pomniejszać przy użyciu scrolla. Mars jest renderowany z realistycznymi teksturami, a wokół niego orbitują jego księżyce – Phobos i Deimos, w tle widoczne są gwiazdy, tworzące efekt kosmicznego otoczenia. W górnej części ekranu znajduje się panel nawigacyjny z przyciskami umożliwiającymi dostęp do różnych funkcji aplikacji, takich jak logowanie, rejestracja, galeria zdjęć Marsa, informacje o burzach piaskowych oraz panele z informacjami o księżycach Marsa. Po wybraniu sektora na powierzchni Marsa, po prawej stronie ekranu pojawia się panel informacyjny z danymi o wybranym sektorze, zawierający opis warunków pogodowych, przycisk Status bazy wyświetlający dodatkowe informacje o bazie marsjańskiej oraz przycisk Zamknij. Galeria zdjęć Marsa jest dostępna po kliknięciu przycisku Galeria, a zdjęcia wyświetlane są w formie siatki, z możliwością powiększenia w lightboxie. Panele logowania i rejestracji pojawiają się po prawej stronie ekranu i zawierają formularze do wprowadzenia danych użytkownika oraz przyciski do zamknięcia panelu. Aplikacja jest w pełni responsywna, dostosowując się do różnych rozmiarów ekranów, a jej kolorystyka utrzymana jest w stonowanych odcieniach czerwieni, szarości i czerni, co nawiązuje do tematyki kosmicznej i planety Mars. Teksty są czytelne, a przyciski i panele mają nowoczesny, minimalistyczny design, co sprawia, że aplikacja jest zarówno funkcjonalna, jak i atrakcyjna wizualnie.

Panel logowania w aplikacji Mars-Dust jest prosty i intuicyjny, zaprojektowany z myślą o łatwym i szybkim dostępie do funkcji dostępnych dla zalogowanych użytkowników. Panel pojawia się po prawej stronie ekranu po kliknięciu przycisku Zaloguj się w panelu nawigacyjnym.



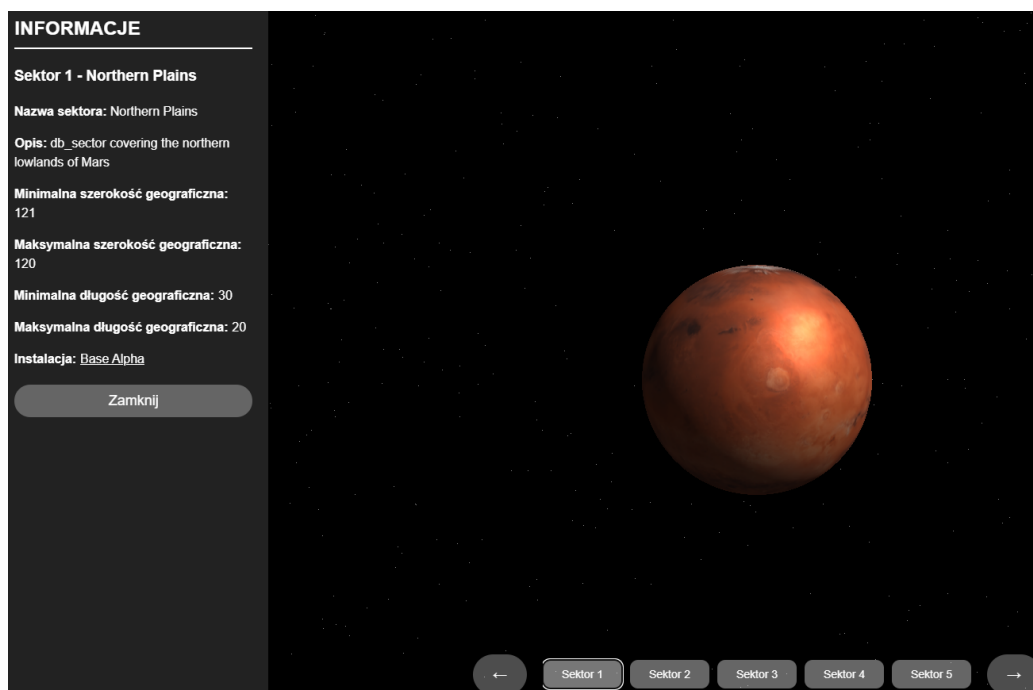
The screenshot shows a dark-themed login panel titled "LOGOWANIE". It contains four input fields: "Nazwa użytkownika" (Username), "Hasło" (Password), and two buttons: "Zaloguj się" (Login) and "Zamknij" (Close). Below the buttons is a link labeled "Panel administratora" (Administrator panel).

Panel rejestracji w aplikacji Mars-Dust jest zaprojektowany w sposób prosty i intuicyjny, umożliwiający użytkownikom łatwe tworzenie nowego konta. Panel pojawia się po prawej stronie ekranu po kliknięciu przycisku Zarejestruj się w panelu nawigacyjnym.

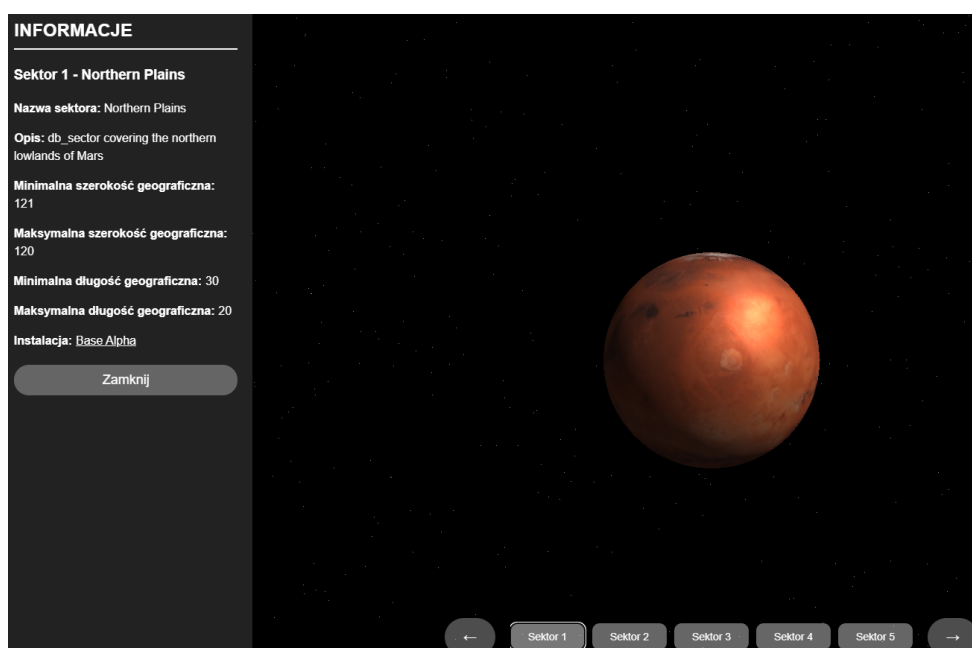


The screenshot shows a dark-themed registration panel titled "REJESTRACJA". It contains five input fields: "Nazwa użytkownika" (Username), "Adres e-mail" (Email address), "Hasło" (Password), "Powtórz hasło" (Repeat password), and two buttons: "Zarejestruj się" (Register) and "Zamknij" (Close).

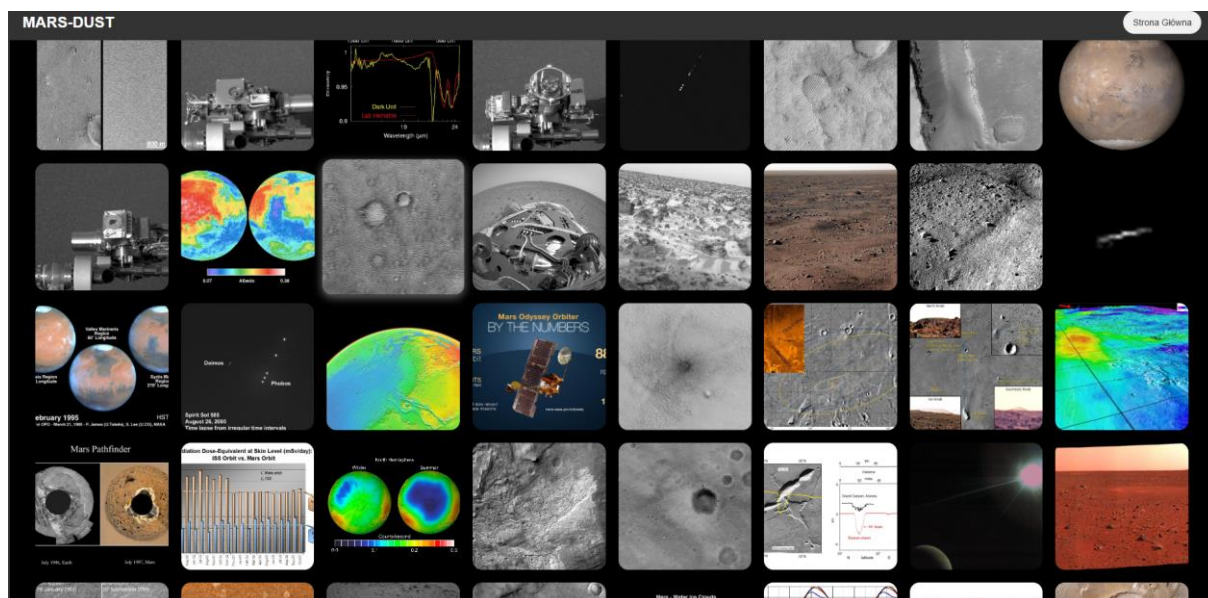
Po zalogowaniu się do aplikacji Mars-Dust, użytkownicy zyskują dostęp do szczegółowych informacji o różnych sektorach na powierzchni Marsa. Funkcja ta pozwala na eksplorację danych dotyczących wybranych obszarów planety, co jest szczególnie przydatne dla osób zainteresowanych badaniem warunków panujących na Marsie.



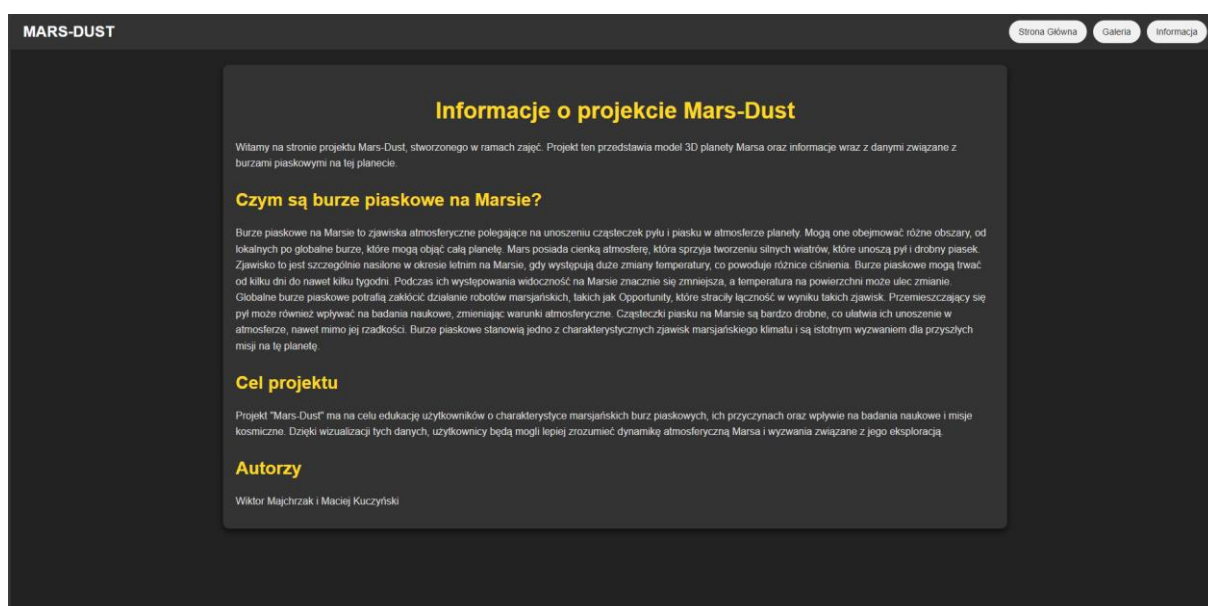
Aplikacja Mars-Dust umożliwia użytkownikom dostęp do informacji o księżycach Marsa – Phobosie i Deimosie – bez konieczności logowania się. Dzięki tej funkcji użytkownicy mogą szybko zapoznać się z danymi dotyczącymi tych naturalnych satelitów planety.



Galeria zdjęć w aplikacji Mars-Dust stanowi sekcję, która pozwala użytkownikom na eksplorację zdjęć Marsa pozyskanych z API NASA. Aplikacja oferuje użytkownikom możliwość podziwiania zdjęć w wysokiej jakości oraz interaktywne wyświetlanie powiększonych obrazów.



Strona „Informacje o projekcie” w aplikacji Mars-Dust dostarcza użytkownikom szczegółowych informacji o projekcie, jego celu oraz przedstawia zagadnienie burz piaskowych na Marsie, a także wpływ tych zjawisk na badania naukowe i przyszłe misje kosmiczne.



W ramach realizacji projektu Mars-Dust zbudowano system zarządzania danymi za pomocą wbudowanego panelu administracyjnego Django, który umożliwia wygodne edytowanie, dodawanie oraz usuwanie różnych danych przechowywanych w aplikacji.

