Programowanie aplikacji internetowych

# Aplikacja internetowa Parku Rozrywki
# Założenia projektowe

Grupa projektowa w składzie:

Adrian Kawczyński
Sylwester Kałucki

Kierunek Telekomunikacja
Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

9 czerwca 2024

# Spis treści

**1. Temat projektu** . . . 2
**2. Zespół i opis ról** . . . 2
**3. Założenia funkcjonalne aplikacji** . . . 2
**4. Założenia architektoniczne aplikacji** . . . 2
**5. Opis wykorzystania narzędzi programowych w powiązaniu z założeniami funkcjonalnymi i architektonicznymi** . . . 3
**6. Opis realizacji** . . . 4
6.1. Proces powstawania aplikacji . . . 4
6.2. Problemy i ich rozwiązania . . . 4
6.3. Modyfikacje rozwiązań i założeń . . . 4
**7. Opis techniczny** . . . 5
7.1. Frontend . . . 5
7.2. Backend . . . 6
7.3. Baza danych . . . 6
**8. Opis instalacji** . . . 6
8.1. Środowisko programistyczne . . . 7
8.2. Baza danych . . . 7
8.3. Tworzenie bazy danych . . . 7
8.4. Uruchomienie aplikacji . . . 7
**9. Opis użytkownika** . . . 7
9.1. Użytkownik niezalogowany . . . 7
9.2. Użytkownik zalogowany . . . 9
9.3. Administrator . . . 11

## 1. Temat projektu

Tematem naszego projektu jest aplikacja internetowa dla parku rozrywki, która będzie obsługiwać trzy perpektywy. Możliwe będzie przeglądanie atrakcji parku rozrywki, zakup lub rezerwacja biletów. W perspektywie administratora możliwe będzie zarządzanie danymi pracowników oraz rezerwacjami biletów.

## 2. Zespół i opis ról

Zespół składał się z dwóch osób:

— Adrian Kawczyński - kierownik projektu, progamista front-end, programista back-end, programista baz danych
— Sylwester Kałucki - progamista front-end, programista back-end, programista baz danych

## 3. Założenia funkcjonalne aplikacji

Aplikacja webowa dla Parku Rozrywki jest zaprojektowana z myślą o trzech perspektywach użytkownika: niezalogowany, użytkownik (USER) oraz administrator (ADMIN).

1. Perspektywa niezalogowanego użytkownika:
   — Dostęp do podstawowych informacji o parku, takich jak dostępne atrakcje i ceny biletów.
   — Przeglądanie atrakcji z możliwością podświetlenia szczegółów po najechaniu kursorem na nazwę atrakcji.
   — Możliwość przeglądania oferty biletowej, jednak bez możliwości dokonania rezerwacji.
2. Perspektywa zalogowanego użytkownika (USER):
   — Dostęp do wszystkich funkcji dostępnych dla niezalogowanego użytkownika.
   — Możliwość rezerwacji biletów poprzez dodawanie ich do koszyka oraz finalizowanie zamówienia.
   — Formularz rezerwacji automatycznie wypełniany informacjami z koszyka, wymagający potwierdzenia użytkownika.
3. Perspektywa administratora (ADMIN):
   — Dostęp do pełnej funkcjonalności użytkownika (USER) oraz dodatkowych funkcji administracyjnych.
   — Możliwość przeglądania, dodawania, edytowania oraz usuwania rekordów z tabel w bazie danych, takich jak pracownicy oraz rezerwacje.
   — Panel administracyjny ze zmienioną kolorystyką interfejsu dla łatwego rozróżnienia od innych perspektyw.

## 4. Założenia architektoniczne aplikacji

Aplikacja wykorzystuje architekturę trójwarstwową Model-View-Controller (MVC), co pozwala na efektywne zarządzanie danymi, logiką biznesową oraz interfejsem użytkownika.

1. **Model**:
   — Reprezentowany przez bazę danych Oracle, która przechowuje wszystkie informacje niezbędne dla aplikacji, takie jak dane użytkowników, pracowników, atrakcji oraz rezerwacji.
   — Użycie Spring JDBC oraz sterownika Java DataBase Connectivity (JDBC) do komunikacji z bazą danych, umożliwiając operacje CRUD na tabelach.
2. **View**:
   — Warstwa widoku składa się z plików HTML oraz szablonów Thymeleaf, które definiują sposób prezentacji danych z modelu.
   — Interfejs użytkownika wzbogacony o CSS i Bootstrap dla poprawy wyglądu oraz JavaScript i jQuery dla interaktywności i obsługi zdarzeń na stronie.
3. **Controller**:
   — Warstwa kontrolera zaimplementowana przy użyciu Spring MVC, obsługuje żądania użytkowników z warstwy widoku, przetwarza je i aktualizuje widoki na podstawie wyników z modelu.

— Kontrolery obsługują logikę logowania, autoryzacji, zarządzania koszykiem, rezerwacji biletów oraz operacji administracyjnych.

## 5. Opis wykorzystania narzędzi programowych w powiązaniu z założeniami funkcjonalnymi i architektonicznymi

1. **Spring Boot:**
   — **Funkcjonalność:** Umożliwia szybkie uruchomienie i automatyczną konfigurację aplikacji webowej.
   — **Architektura:** Integruje różne moduły Spring (MVC, JDBC) w jedną, łatwą do zarządzania całość, wspierając architekturę trójwarstwową (Model-View-Controller).
2. **Spring MVC:**
   — **Funkcjonalność:** Zarządza przepływem danych i interakcjami użytkownika w aplikacji. Przetwarza żądania HTTP, mapuje je do odpowiednich metod kontrolera i generuje odpowiedzi.
   — **Architektura:** Implementuje wzorzec MVC, umożliwiając oddzielenie logiki biznesowej (kontroler), warstwy danych (model) i interfejsu użytkownika (widok).
3. **Spring JDBC:**
   — **Funkcjonalność:** Umożliwia wykonywanie operacji CRUD na bazie danych Oracle, co jest kluczowe dla zarządzania danymi użytkowników, pracowników, rezerwacji i atrakcji.
   — **Architektura:** Działa w warstwie modelu, zapewniając komunikację z relacyjną bazą danych i umożliwiając kontrolerom dostęp do danych.
4. **Spring Thymeleaf:**
   — **Funkcjonalność:** Generuje dynamiczne strony HTML, wyświetlając dane pobrane z modelu.
   — **Architektura:** Umożliwia tworzenie warstwy widoku, która jest integralną częścią wzorca MVC, łącząc dane z kontrolera z odpowiednimi szablonami HTML.
5. **Java:**
   — **Funkcjonalność:** Implementuje logikę biznesową aplikacji, zarządzając funkcjonalnościami użytkownika (rejestracja, logowanie, rezerwacje) oraz administrowaniem danymi.
   — **Architektura:** Wspiera wszystkie warstwy aplikacji: kontrolery, warstwę modelu (logika biznesowa) oraz integrację z widokami poprzez Spring MVC i Thymeleaf.
6. **IntelliJ IDEA:**
   — **Funkcjonalność:** Zapewnia wsparcie dla programistów poprzez funkcje takie jak automatyczne wykrywanie błędów, podpowiedzi kodu, debugowanie i zarządzanie zależnościami.
   — **Architektura:** Ułatwia zarządzanie i rozwój aplikacji, wspierając utrzymanie struktury kodu zgodnej z architekturą MVC.
7. **GitHub:**
   — **Funkcjonalność:** Umożliwia hostowanie, zarządzanie wersjami oraz współdzielenie kodu źródłowego projektu.
   — **Architektura:** Wspiera współpracę zespołową, zapewniając spójność kodu i łatwe śledzenie zmian, co jest kluczowe dla utrzymania architektury aplikacji.
8. **HTML, CSS, Bootstrap:**
   — **Funkcjonalność:** Budują strukturę i stylizację interfejsu użytkownika, zapewniając responsywny i nowoczesny wygląd aplikacji.
   — **Architektura:** Umożliwiają tworzenie warstwy widoku, która jest integralną częścią wzorca MVC, wspierając dynamiczne generowanie stron przez Thymeleaf.
9. **JavaScript i jQuery:**
   — **Funkcjonalność:** Dodają interaktywność i obsługę zdarzeń na stronie, umożliwiając dynamiczne manipulowanie DOM i poprawę doświadczenia użytkownika.
   — **Architektura:** Wspierają warstwę widoku, umożliwiając interakcję użytkownika z elementami HTML wygenerowanymi przez Thymeleaf.
10. **Oracle Database i Oracle SQL Developer:**
    — **Funkcjonalność:** Przechowują i zarządzają danymi aplikacji, takimi jak informacje o użytkownikach, pracownikach, atrakcjach i rezerwacjach.
    — **Architektura:** Stanowią warstwę modelu, zapewniając trwałe przechowywanie danych i umożliwiając dostęp do nich poprzez Spring JDBC, wspierając w ten sposób logikę biznesową aplikacji.

# 6. Opis realizacji

## 6.1. Proces powstawania aplikacji

Proces tworzenia aplikacji składał się z kilku etapów, od analizy wymagań po wdrożenie i testowanie. Poniżej przedstawiono kluczowe etapy realizacji projektu:

1. **Analiza wymagań:**
   - Zidentyfikowano potrzeby użytkowników i funkcjonalności, które aplikacja miała spełniać.
   - Wymagania te obejmowały dostępność dwóch perspektyw (administrator i użytkownik) oraz określone operacje, takie jak rezerwacja biletów czy zarządzanie danymi pracowników.
2. **Projektowanie architektury:**
   - Zdecydowano się na wykorzystanie wzorca architektonicznego Model-View-Controller (MVC), co pozwoliło na oddzielenie logiki biznesowej, interfejsu użytkownika oraz zarządzania danymi.
   - Przygotowano diagramy architektury systemu, uwzględniające podział na warstwy oraz interakcje między nimi.
3. **Implementacja backendu:**
   - Rozpoczęto od konfiguracji projektu w Spring Boot, co umożliwiło szybkie uruchomienie aplikacji.
   - Zaimplementowano kontrolery w ramach Spring MVC, które odpowiadały za obsługę żądań HTTP i komunikację z bazą danych przy użyciu Spring JDBC.
4. **Implementacja frontendu:**
   - Użyto Thymeleaf do generowania dynamicznych stron HTML.
   - Zastosowano HTML, CSS oraz Bootstrap do stworzenia responsywnego i przyjaznego interfejsu użytkownika.
   - Wykorzystano JavaScript i jQuery do dodania interaktywności i dynamicznego zarządzania zawartością stron.
5. **Integracja z bazą danych:**
   - Skonfigurowano połączenie z bazą danych Oracle.
   - Utworzono schematy i tabele w bazie danych, które przechowywały dane aplikacji, takie jak informacje o użytkownikach, pracownikach, atrakcjach i rezerwacjach.
6. **Testowanie:**
   - Testowano interfejs użytkownika pod kątem responsywności oraz poprawności działania różnych funkcji.

## 6.2. Problemy i ich rozwiązania

Podczas realizacji projektu napotkano kilka problemów, które wymagały szczególnej uwagi:

- **Problemy z konfiguracją połączenia z bazą danych:**
  - *Opis problemu:* Początkowo występowały trudności z poprawną konfiguracją połączenia do bazy danych Oracle, co uniemożliwiało wykonywanie zapytań.
  - *Rozwiązanie:* Dokładnie przeanalizowano dokumentację oraz przykładowe konfiguracje, co pozwoliło na poprawne ustawienie parametrów połączenia w pliku konfiguracyjnym aplikacji.
- **Niekompatybilność wersji bibliotek:**
  - *Opis problemu:* Występowały konflikty między wersjami używanych bibliotek (np. Spring Boot, Thymeleaf, JDBC), co powodowało błędy kompilacji.
  - *Rozwiązanie:* Przeprowadzono analizę zależności i zaktualizowano wersje bibliotek do kompatybilnych ze sobą, co pozwoliło na wyeliminowanie konfliktów.
- **Problem z pobieraniem danych z bazy:**
  - *Opis problemu:* Podczas prób odczytywania danych o atrakcjach z bazy danych występowały błędy, z których nie dało się bezpośrednio zidentyfikować problemu
  - *Rozwiązanie:* Należało zwiększyć limit znaków w przypadku kolumny "opis"

## 6.3. Modyfikacje rozwiązań i założeń

W trakcie realizacji projektu konieczne było wprowadzenie pewnych modyfikacji w stosunku do pierwotnych założeń:

— **Rozbudowa funkcjonalności panelu administratora:**
  — *Modyfikacja:* Pierwotnie panel administratora miał umożliwiać jedynie podstawowe operacje na danych. Zdecydowano się na dodanie zaawansowanych funkcji, takich jak sortowanie tabel czy automatyczne podświetlanie wierszy.
  — *Powód:* Umożliwienie bardziej efektywnego zarządzania danymi oraz poprawa użyteczności panelu administratora.
— **Ulepszenie interfejsu użytkownika:**
  — *Modyfikacja:* Zdecydowano się na zastosowanie dodatkowych bibliotek, takich jak Bootstrap, aby zapewnić lepszą responsywność i estetykę interfejsu.
  — *Powód:* Zwiększenie atrakcyjności wizualnej aplikacji oraz poprawa doświadczenia użytkownika.
— **Wprowadzenie obsługi błędów:**
  — *Modyfikacja:* Dodano dedykowane strony błędów (np. 404) oraz mechanizmy wyłapywania i raportowania błędów.
  — *Powód:* Zapewnienie lepszej komunikacji z użytkownikiem w przypadku wystąpienia błędów oraz ułatwienie diagnozowania problemów.

# 7. Opis techniczny

## 7.1. Frontend

Część frontend aplikacji składa się z zasobów statycznych jak i szablonów HTML, które tworzą interfejs użytkownika.
Szablony HTML tworzą zarys strony internetowej, której wygląd następnie był stylizowany za pomocą plików z rozszerzeniem ".cs".
Pliki napisane w języku JavaScript służyły do dodania funkcjonalności na danej stronie internetowej np. pracownicy.js implementuje sortowanie tabeli po kliknięciu w nazwę kolumny oraz podświetlanie wierszy.

```
const getCellValue = (tr, idx) => tr.children[idx].innerText || tr.children[idx].textContent;  Show usages

const comparer = (idx, asc) => (a, b) => ((v1, v2) =>  Show usages
        v1 !== '' && v2 !== '' && !isNaN(v1) && !isNaN(v2) ? v1 - v2 : v1.toString().localeCompare(v2)
)(getCellValue( tr: asc ? a : b, idx), getCellValue( tr: asc ? b : a, idx));

document.querySelectorAll( selectors: 'th').forEach( callbackfn: th : HTMLTableCellElement  => th.addEventListener( type: 'click', (() : void  => {
    const table : HTMLTableElement  = th.closest( selector: 'table');
    const tbody : HTMLTableSectionElement  = table.querySelector( selectors: 'tbody');
    Array.from(tbody.querySelectorAll( selectors: 'tr'))
        .sort(comparer(Array.from(th.parentNode.children).indexOf(th),  asc: this.asc = !this.asc))
        .forEach(tr : HTMLTableRowElement  => tbody.appendChild(tr) );})));


function highlightRows() : void  {  Show usages
    const iloscWierszy : jQuery  = $("#tabela tr").length;
    for (var i : number  = 0; i < iloscWierszy+1; i++){
        if ($("table tr").eq(i).find("td").eq(5).text() === "TAK") {
            $("table tr").eq(i).addClass("active")
        }
    }
}
highlightRows();
```

Rysunek 1: Kod pracownicy.js

### 7.2. Backend

Backend aplikacji tworzą pliki napisane w języku Javaz znajdujace się w katalogu DAO. Poniżej kod przykładowego pliku Atrakcje.java:

```java
package PAINT_projekt.SpringApplication;

public class Atrakcje {  20 usages



    private int id;  4 usages

    private String nazwa;  4 usages



    private int data_przegladu;  4 usages
    private String opis;  4 usages

    private String link;  4 usages

    public Atrakcje(){}  1 usage
    public int getId() { return id; }

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

    public String getNazwa() { return nazwa; }

    public void setNazwa(String nazwa) { this.nazwa = nazwa; }

    public int getData_przegladu() { return data_przegladu; }

    public void setData_przegladu(int data_przegladu) { this.data_przegladu = data_przegladu; }

    public String getOpis() { return opis; }

    public void setOpis(String opis) { this.opis = opis; }

    public String getLink() { return link; }

    public void setLink(String link) { this.link = link; }



    public Atrakcje(int id, String nazwa, int data_przegladu, String opis, String link) {  no usages
        this.id = id;
        this.nazwa = nazwa;
        this.data_przegladu = data_przegladu;
        this.opis = opis;
        this.link = link;
    }
    @Override
    public String toString() {
        return "Atrakcje{" +
                "id=" + id +
                ", nazwa='" + nazwa + '\'' +
                ", data_przegladu=" + data_przegladu +
                ", opis='" + opis + '\'' +
                ", link='" + link + '\'' +
                '}';
    }
```

Rysunek 2: Kod Atrakcje.java

Dzięki temu plikowi możliwa jest komunikacja między aplikacją webową, a bazą danych. Odpowiada on za odczytywanie wartości z bazy danych jak i modyfikacje lub dodawanie nowych wpisów.

### 7.3. Baza danych

Użyta baza danych jest bazą danych firmy Oracle. Jest to relacyjna baza danych, co oznacza, że dane są przechowywane w tabelach, które mogą być ze sobą powiązane za pomocą kluczy obcych. Relacyjny model danych ułatwia organizację, zarządzanie i dostęp do danych, umożliwiając tworzenie złożonych zapytań SQL, które mogą łączyć dane z różnych tabel. Komunikacja między aplikacją z bazą danych umożliwia interfejs API - JDBC (Java Databate Connectivity).

## 8. Opis instalacji

Poprawne uruchomienie projektu wymaga zainstalowania:

— Środowiska programistycznego wraz z Java JDK - w naszym przypadku było to IntelliJ IDEA i Oracle Java JDK 17
— Bazy danych Oracle wraz z interfejsem graficznym ułatwiającym tworzenie baz danych

### 8.1. Środowisko programistyczne

Do wykonania projektu skorzystano z środowiska programistycznego IntelliJ IDEA. Link do pobrania: IntelliJ IDEA
Dodatkowo należy zainstalować Java'e. Do projektu użyto wersjiOracle Java JDK 17.

### 8.2. Baza danych

Baza danych użyta w projekcie to nOracle Database 21c Express Editio.
Bazę danych należy zainstalowano korzystając z ustawień domyślnych.
Do bazy danych doinstalowano narzędzie graficzne ułatwiające pracę z bazą danych, SQL Developer. Tą aplikacje również zainstalowano według domyślnych ustawień.

### 8.3. Tworzenie bazy danych

Po uruchomieniu oprogramowania SQL Developer, należy stworzyć nową bazę danych klikając w "New/ Select Database Connection", nadając mu dowolną nazwę oraz wykorzystując hasło użyte podczas instalacji bazy danych. Następnie należy w nowo stworzonym połączeniu utworzyć nowego użytkownika. Ważne, aby tworzony użytkownik miał wybrane parametry "DBA" (zakładka Granted Roles), "ALTER SESSION" (zakładka System Privilages) oraz "USERS" (zakładka Quotas). Ostatnim krokiem jest zalogowanie się do bazy danych oraz skopiowanie dostarczonego kodu SQL w zakładce "SQL Worksheet".

### 8.4. Uruchomienie aplikacji

Przed uruchomieniem aplikacji w aplikacji IntelliJ IDEA należy zmienić parametry w pliku "application.properties" na te użyte podczas tworzenia nowego użytkownika bazy danych.

```
server.error.whitelabel.enabled=false
spring.datasource.url=jdbc:oracle:thin:@127.0.0.1:1521:xe
spring.datasource.username=AKAWCZ4
spring.datasource.password=ADMIN
logging.level.root=INFO
```

Rysunek 3: kod "appliaction.properties"

Uruchomienie aplikacji odbywa się poprzez uruchomienie pliku "Application.java". Następnie należy połączyć się z odpowiednią witryną. Domyślnie jest to localhost.
Dodatkowo, należy pamiętać, aby przed uruchomieniem aplikacji, zalogować się do bazy danych za pomocą aplikacji SQL Developer.

## 9. Opis użytkownika

Aplikacja internetowa ma trzy perspektywy: osoby niezalogowanej, użytkownika zalogowanego oraz administratora. Nawigacja między stronami odbywa się za pomocą paska zakładek zlokalizowanego na górze strony internetowej.

### 9.1. Użytkownik niezalogowany

Domyślną perspektywą jest perspektywa osoby niezalogowanej. Ta perspektywa posiada dostęp do takich podstron jak strona główna, "bilety" i "atrakcje". Główna strona składa się z paska zakładek, oraz głownego panelu zawierającego przycisk przekierowujący użytkownika aplikacji na oddzielną stronę logowania.

Rysunek 4: Strona główna użytkownika niezalogowanego

Podstrony "bilety" zawiera informacje o wszystkich rodzajach biletów dostępnych w parku rozrywki. Ich rezerwacja jest niemożliwa, wymaga zalogowania się.

Rysunek 5: Wygląd strony z koszykiem przed zalogowaniem.

Podstrona "atrakcje" zawiera informacje o wszystkich atrakcjach oferowanych na terenie parku rozrywki. Możliwe jest wyświetlenie informacji o interesującej przez nas atrakcji poprzez najechanie kursorem na daną nazwę. Zostanie wtedy wyświetlony obraz przedstawiający atrakcję, krótki opis oraz kropka z lokalizacją atrakcji zmieni kolor z żółtego na czerwony. Przeglądanie atrakcji może się również odbywać poprzez wybór kropki na mapie.

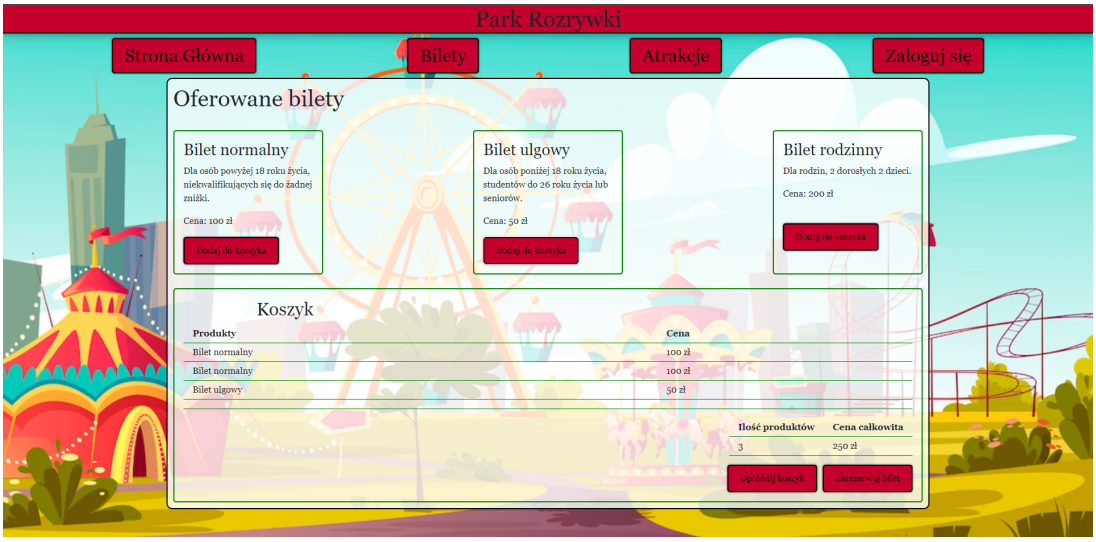
Rysunek 6: Strona przedstawiająca atrakcje dostępne w parku.

Perspektywa osoby niezalogowanej ma również dostęp do panelu logowania.

Rysunek 7: Wygląd strony logowania.

### 9.2. Użytkownik zalogowany

W celu zalogowania się jako "user" konieczne jest wpisanie "user" zarówno w polu login jak i hasło. Użytkownik zalogowany posiada jedną dodatkową możliwość w porównaniu do osoby niezalogowanej. Na podstronie "bilety" posiada on możliwość dodawania inetersujących go biletów do koszyka.




Rysunek 8: Wygląd koszyka po zalogowaniu na konto usera.

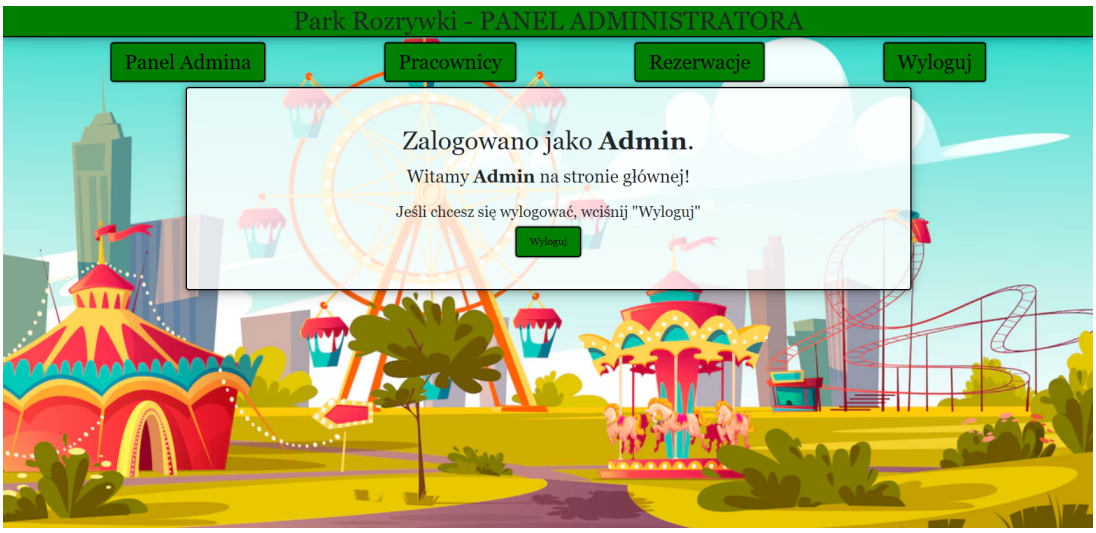
Następnie może wybrane przez siebie bilety zarezerwować. Ta czynność wymaga wprowadzenia niezbędnych danych do pomyślnej rezerwacji.

Rysunek 9: Formularz rezerwacji.

Złożone rezerwacje, później będą widoczne w widoku administratora.

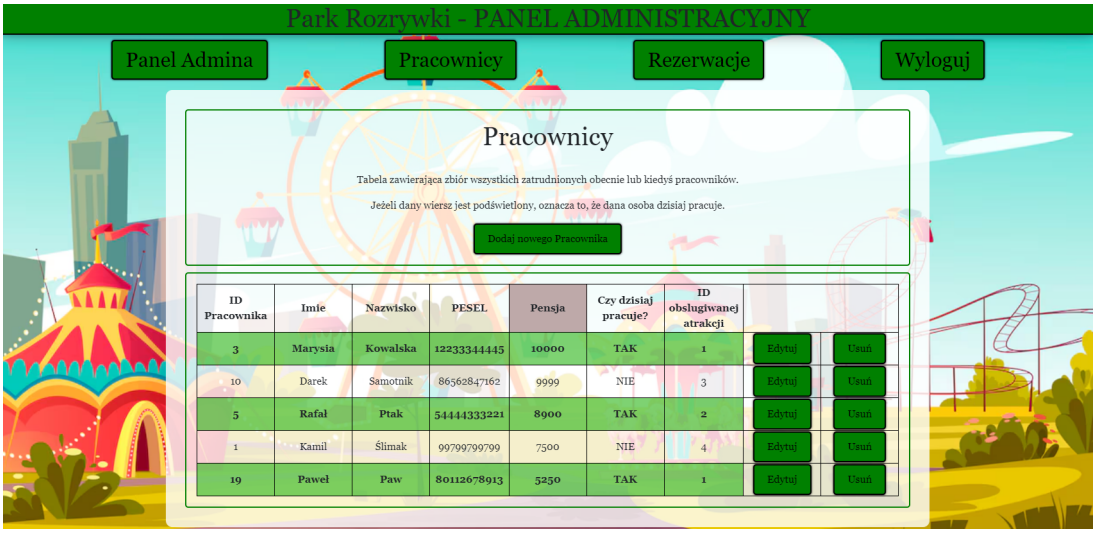
### 9.3. Administrator

W celu zalogowania się jako administrator konieczne jest wpisanie "admin" zarówno w polu login jak i hasło.
Perspektywa administratora oferuje dostęp do takich podstron jak panel główny administratora, "pracownicy" oraz "rezerwacje". Po zalogowaniu się jako "admin", strona zmienia swój kolor na zielony. Główny panel informuje nas o pomyślnym zalogowaniu oraz daje opcję wylogowania się.



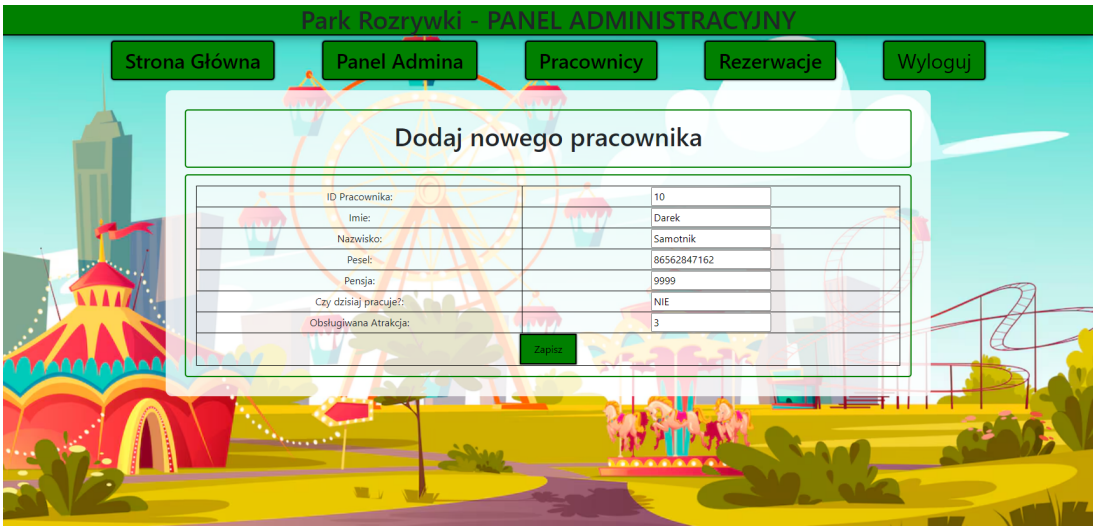

Rysunek 10: Wygląd panelu administratora.

Podstrona "pracownicy" zawiera tabelę z wszystkimi pracownikami parku rozrywki. Administrator ma możliwość usunięcia pracownika, edycji jego danych lub dodania nowego.




Rysunek 11: Wygląd strony ze wszystkimi pracownikami.

Po kliknięciu w opcje "Dodaj nowego pracownika", następuje przekierowanie na stronę wypełniania danych nowego pracownika.




Rysunek 12: Wygląd strony z dodawaniem pracownika.

Po kliknięciu w "edytuj", również nastąpi przekierowanie. Strona ta jest bardzo podobna w wyglądzie do strony dodawania nowego pracownika, z tym, że dane zostaną wypełnione.

Po poprawnym wpisaniu wszystkich wartości oraz zatwierdzeniu, w tabeli pojawi się wiersz z jej danymi.

W celu usunięcia informacji o pracowniku należy kliknąć w "Usuń" w jego wierszu.

Dodatkowo tabela "pracownicy" podświetla wiersze osób pracujących danego dnia oraz ma możliwość sortowania poprzez kliknięcie na tytuł danej kolumny.

Podstrona "rezerwacje" zawiera informacje o wszystkich dokonanych rezerwacjach. Tutaj zbierane są wszystkie rezerwacje złożone w perspektywie osoby zalogowanej.

Rysunek 13: Wygląd strony z rezerwacjami.

Tabela ma takie same funkcjonalności jak ta użyta na podstronie "pracownicy".