

# **Rozproszone Systemy Informatyczne**

Raport - MiniProjekt

Paweł Kluska, 260391

Katsiaryna Ziatsikava, 245891

# Stos technologiczny aplikacji

- Frontend - Vue.js
- Backend - Spring boot
- Baza danych - Postgresql
- Hosting - Amazon Web Services

## Baza danych

Baza danych składa się z 2 tabel połączonych relacją 1 - n. Są to tabela equipment oraz category. (Kategoria może mieć wiele sprzętów, sprzęt jedną kategorię).

Schemat bazy prezentuje się następująco:

```
create table if not exists category
(
    id    bigserial
        primary key,
    name  varchar(255)
);

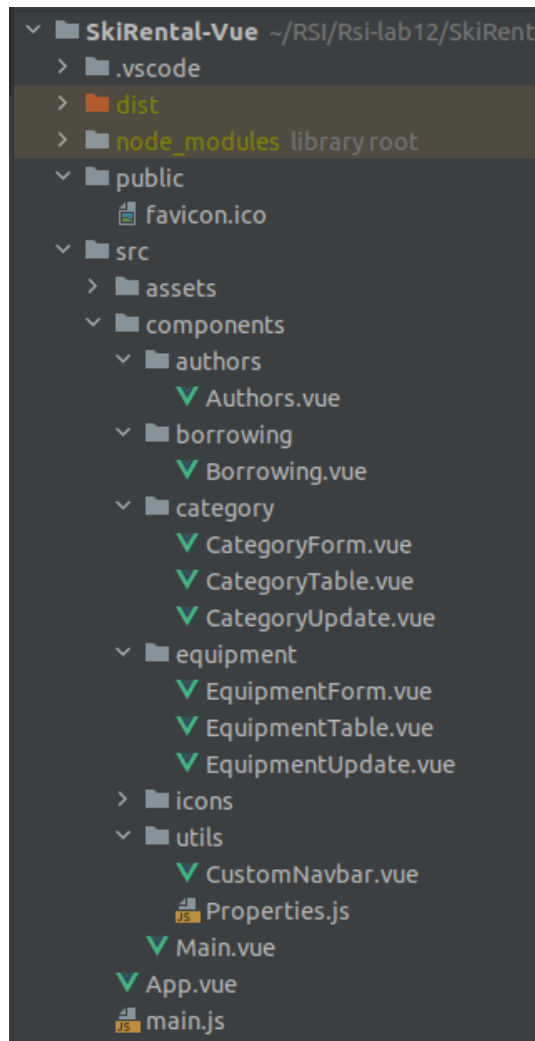
create table if not exists equipment
(
    id            bigserial
        primary key,
    description   varchar(255),
    image         varchar(255),
    is_borrowed  boolean          not null,
    name          varchar(255),
    price         double precision not null
        constraint equipment_price_check
            check (price >= (0)::double precision),
    size          varchar(255),
    category_id  bigint           not null
        constraint fkb4svnpl9s3b1p5uuvqx5ihlki
            references category
);
```

## Główne kroki projektowania frontendu

Projekt składa się z 4 części:

- Tabeli sprzętu
- Tabeli kategorii
- Zakładki dotyczącej wypożyczeń
- Sekcji autorów

Cała struktura projektu prezentuje się następująco:



Do każdego z komponentów został zaimportowany navbar znajdujący się w pliku CustomNavbar.vue, dzięki temu można łatwo nawigować po aplikacji

Tabele kategorii oraz sprzętu są niemal identyczne do siebie, dlatego zostanie omówiona tylko jedna z nich.

## Tabela sprzętu

Na sekcję dotyczącą tabeli sprzętu składają się 3 komponenty - tabela wyświetlająca dane, formularz do dodawania nowego sprzętu oraz formularz do aktualizacji istniejącego sprzętu.

Formularz dotyczący dodawania sprzętu został zaimportowany do tabeli, przez co można je wyświetlić razem na jednej stronie.

## Formularz dodawania sprzętu

```
<template>
  <div id="person-form" class="container">
```

```
<h1>Dodaj nowy sprzęt</h1>
<form @submit.prevent="handleSubmit">
  <label>Nazwa</label>
  <input
    v-model="equipment.name"
    type="text"
    :class="{ 'has-error': submitting && invalidName}"
    @focus="clearStatus"
    @keypress="clearStatus"
  />
  <p v-if="submitting && invalidName" class="error-message">
    Nazwa nie może być pusta
  </p>

  <label>Rozmiar</label>
  <input
    v-model="equipment.size"
    type="text"
    :class="{ 'has-error': submitting && invalidSize}"
    @focus="clearStatus"
    @keypress="clearStatus"
  />
  <p v-if="submitting && invalidSize" class="error-message">
    Rozmiar nie może być pusty
  </p>

  <label>Cena</label>
  <input
    v-model="equipment.price"
    type="number" step="0.01"
    :class="{ 'has-error': submitting && invalidPrice}"
    @focus="clearStatus"
    @keypress="clearStatus"
  />
  <p v-if="submitting && invalidPrice" class="error-message">
    Cena musi być liczbą większą od 0.
  </p>

  <label>Opis</label>
  <input
    v-model="equipment.description"
    type="text"
    @focus="clearStatus"
    @keypress="clearStatus"
  />
```

```

<label>Zdjęcie</label>
<input
  v-model="equipment.image"
  type="text"
  :class="{ 'has-error': submitting && invalidImage}"
  @focus="clearStatus"
  @keypress="clearStatus"
/>
<p v-if="submitting && invalidImage" class="error-message">
  Proszę podać link do zdjęcia.
</p>

<label>Kategoria</label>
<select v-model="equipment.category"
  :class="{ 'has-error': submitting && invalidCategory}">
  <option
    v-for="c in categories" :key="c.id"
    @focus="clearStatus"
  >
    {{ c.name }}
  </option>
</select>
<p v-if="submitting && invalidCategory" class="error-message">
  Proszę wybrać kategorię.
</p>

<p v-if="error && submitting" class="error-message">
  Proszę wypełnić wskazane pola formularza.
</p>
<p v-if="success" class="success-message">
  Dane poprawnie zapisano.
</p>
<button class="btn btn-primary mt-4">Dodaj sprzęt</button>
</form>
</div>
</template>

```

Został tutaj zdefiniowany formularz wraz z wszystkimi potrzebnymi polami. Dodatkowo, każde z pól jest walidowane, przez co nie można przesłać na serwer nieprawidłowych danych.

```

import axios from "axios";
import { apiLocation } from "@/components/utils/Properties";

export default {
  name: 'EquipmentForm',
  props: {

```

```

    equipmentsSource: Array,
  },
  data() {
    return {
      submitting: false,
      error: false,
      success: false,
      equipment: {
        name: '',
        size: '',
        price: '',
        description: '',
        image: '',
        isBorrowed: '',
        category: '',
      },
      categories: [],
    }
  },
  mounted() {
    this.getCategories()
  },
  methods: {

    async getCategories() {
      try {
        const response = await fetch(`${apiLocation}/get/categories`)
        const data = await response.json()
        this.categories = data
      } catch (error) {
        console.error(error)
      }
    },

    async postData() {
      try {
        const category = this.categories.filter(obj => {
          return obj.name === this.equipment.category
        });
        const response = await axios.post(`${apiLocation}/post/equipment`, {
          name: this.equipment.name,
          size: this.equipment.size,
          price: this.equipment.price,
          description: this.equipment.description,
          image: this.equipment.image,
          borrowed: false,

```

```

        category: category[0]
    })
    console.log(response.data)
    const savedEquipment = await fetch(`${apiLocation}/get/equipments`)
    const equipments = await savedEquipment.json()
    this.equipmentsSource.push(equipments[equipments.length - 1])
  } catch (error) {
    console.error(error)
  }
},

handleSubmit() {
  this.submitting = true
  this.clearStatus()

  if (this.invalidName || this.invalidSize || this.invalidPrice ||
this.invalidImage || this.invalidCategory) {
    this.error = true
    return
  }
  this.postData()

  this.equipment = {
    name: '',
    size: '',
    price: '',
    description: '',
    image: '',
    isBorrowed: '',
    category: '',
  }
  this.error = false
  this.success = true
  this.submitting = false
},
clearStatus() {
  this.success = false
  this.error = false
},
},
computed: {
  invalidName() {
    return this.equipment.name === ''
  },

  invalidSize() {

```

```

        return this.equipment.size === ''
    },

    invalidPrice() {
        return this.equipment.price < 0 || this.equipment.price === ''
    },

    invalidImage() {
        return this.equipment.image === ''
    },

    invalidCategory() {
        return this.equipment.category === null || this.equipment.category === ''
    },

    },
}
</script>

```

Zostały stworzone metody walidujące każde z pól formularza, metoda przetwarzająca wysłanie żądania dodania sprzętu oraz metoda pobierająca informacje o kategoriach (kategorie można wybierać przy pomocy elementu select). Do formularza dodano kilka klas css w celu poprawienia wizerunku strony.

```

<style scoped>
form {
    margin: 0 auto;
}

[class*='-message'] {
    font-weight: 500;
}

.error-message {
    color: #d33c40;
}

.success-message {
    color: #32a95d;
}

label {
    display: block;
    margin-bottom: 0.5rem;
}
</style>

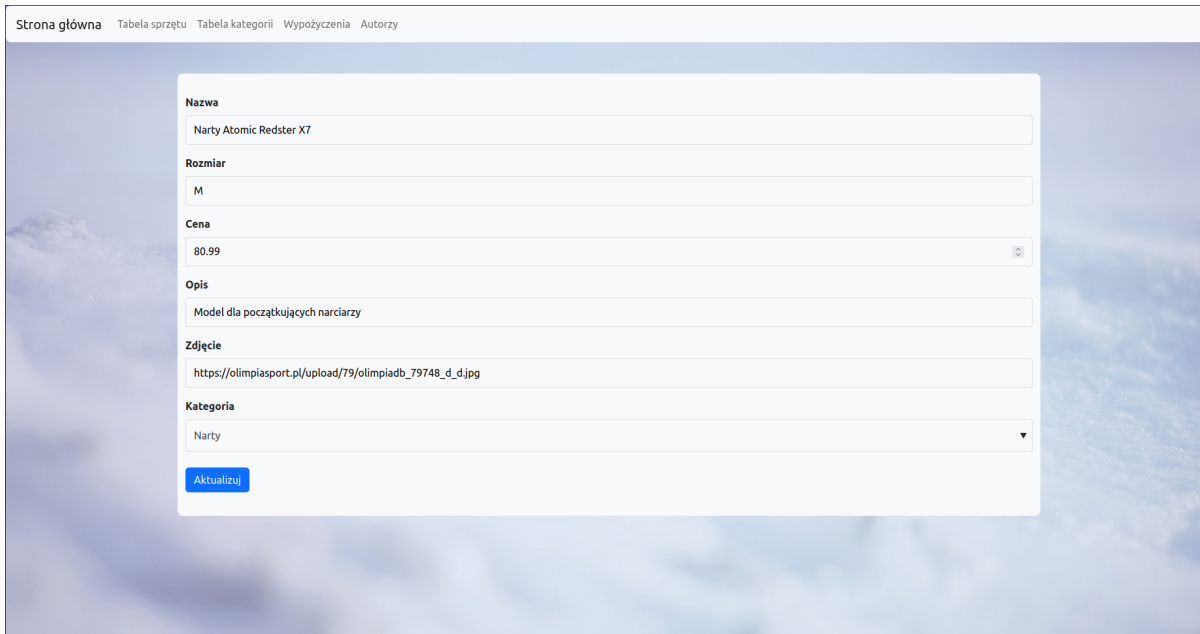
```



## Formularz aktualizacji sprzętu

Ma on dokładnie taką samą strukturę jak poprzedni formularz, z tą różnicą że aktualne dane edytowanego sprzętu są od razu ładowane i wyświetlone w formularzu. Dzięki temu nie trzeba od nowa wpisywać wszystkich danych, jeśli chce się tylko zmienić jeden atrybut. Dodatkowo zamiast metody Post jest wysyłany Patch.

Wygląd:



The screenshot shows a web application interface with a navigation bar at the top containing links: Strona główna, Tabela sprzętu, Tabela kategorii, Wypożyczenia, and Autorzy. The main content area features a form for updating equipment details. The form fields are as follows:

- Nazwa:** Narty Atomic Redster X7
- Rozmiar:** M
- Cena:** 80.99
- Opis:** Model dla początkujących narciarzy
- Zdjęcie:** https://olimpiasport.pl/upload/79/olimpiadb\_79748\_d.jpg
- Kategoria:** Narty (selected from a dropdown menu)

At the bottom of the form is a blue button labeled "Aktualizuj".

## Tabela sprzętu

W tym komponencie trzeba było wysłać żądania do backendu w celu uzyskania informacji o istniejącym sprzęcie i wyświetlić dane w tabeli.

```
<script>
import axios from "axios";
import CategoryForm from "@/components/category/CategoryForm.vue";
import CustomNavbar from "@/components/utils/CustomNavbar.vue";
import {apiLocation} from "@/components/utils/Properties";

export default {
  name: 'CategoryTable',
  components: {CustomNavbar, CategoryForm},

  data() {
    return {
      categories: [],
      currentPage: 1,
    }
  }
}
```

```

    itemsPerPage: 3,
    serverAvailable: true,
  },
},

mounted() {
  this.getCategories()
},

methods: {
  paginatedCategories() {
    const startIndex = (this.currentPage - 1) * this.itemsPerPage;
    const endIndex = startIndex + this.itemsPerPage;
    let slicedEquipments = this.categories.slice(startIndex, endIndex);
    if (slicedEquipments.length === 0 && this.currentPage > 1) {
      this.currentPage--;
    }
    while (slicedEquipments.length < this.itemsPerPage) {
      slicedEquipments.push({
        id: '',
        name: '',
      });
    }
    return slicedEquipments;
  },
  async getCategories() {
    try {
      const response = await fetch(`${apiLocation}/get/categories`)
      const data = await response.json()
      this.categories = data
      this.categories.sort((a, b) => (a.id > b.id) ? 1 : -1)
      this.serverAvailable = true;
    } catch (error) {
      console.error(error)
      this.serverAvailable = false;
    }
  },
  async deleteCategory(itemId) {
    axios.delete(`${apiLocation}/delete/category/${itemId}`)
      .then(response => {
        console.log(response.data);
        this.categories = this.categories.filter(obj => {
          return obj.id !== itemId
        });
      })
      .catch(error => {

```

```

        alert("Nie można usunąć kategorii, ponieważ jest przypisana do
sprzętu!")
        console.error(error);
    });
}
},
computed: {
    totalPages() {
        return Math.ceil(this.categories.length / this.itemsPerPage);
    }
},
}
</script>

```

Są tutaj zdefiniowane metody, które wysyłają odpowiednie requesty.

Gdy dane zostaną załadowane, są one wyświetlone na stronie.

```

<template>
  <CustomNavbar></CustomNavbar>
  <div id="categories-table" class="container bg-light pt-3 mt-5 mb-5 pb-3
rounded-3">
    <h4 v-if="!this.serverAvailable" class="error-message">
      Nie udało się nawiązać połączenia z serwerem
    </h4>

    <table id="table">
      <thead>
        <tr class="header">
          <th>Nazwa</th>
          <th>Usuń</th>
          <th>Aktualizuj</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="(category, index) in paginatedCategories()"
:key="category.id">
          <td>{{ category.name }}</td>
          <td>
            <button class="btn btn-danger" v-if="category.id !== ''"
@click="deleteCategory(category.id)">Usuń</button>
          </td>
          <td>
            <button class="btn btn-primary " v-if="category.id !== ''"

```

```

        @click="this.$router.push({ name: 'categoryUpdate', params: {
id: category.id } });">Aktualizuj
      </button>
    </td>
  </tr>
</tbody>
</table>

<div class="pagination">
  <button class="btn btn-primary" :disabled="currentPage === 1"
@click="currentPage--">Prev</button>
  <span class="ms-3 me-3">Page {{ currentPage }} of {{ totalPages }}</span>
  <button class="btn btn-primary" :disabled="currentPage === totalPages"
@click="currentPage++">Next</button>
</div>
<CategoryForm :categoriesSource="categories" class="mt-5"/>
</div>
</template>

```

Są one wyświetlane przy pomocy pętli dostarczanej przez Vue, która iteruje po załadowanej wcześniej kolekcji danych o sprzęcie. Są tutaj również zastosowane style dostarczane przez bibliotekę Bootstrap, oraz style własnoręcznie zdefiniowane. Na dole strony został załadowany formularz dodawania elementu.

```

</script>

<style scoped>

tr {
  height: 60px;
}

.header {
  height: 50px;
}

.error-message {
  color: #d33c40;
}
</style>

```

Wygląd:

Strona główna

Tabela sprzętuTabela kategoriiWypożyczeniaAutorzy

Nazwa	Rozmiar	Cena	Opis	Zdjęcie	Kategoria	Czy wypożyczona	Usuń	Aktualizuj
Narty Atomic Redster X7	M	80.99	Model dla początkujących narciarzy	https://olimpiasport.pl/upload/79/olimpiadb_79748_d_d.jpg	Narty	false	Usuń	Aktualizuj
Snowboard Burton Custom	L	65.5	Model dla zaawansowanych snowboardzistów	https://snowboardok.pl/21149-large_default/p1530-snowboard-jones-wairheart.jpg	Snowboard	false	Usuń	Aktualizuj
Kask narciarski POC Obex Spin	S	23.8	Lekki wygodny kask dla początkujących	https://deshop.pro/wp-content/uploads/2020/09/30B4694_E533_A.jpg	Kaski	false	Usuń	Aktualizuj
Gogle narciarskie Oakley Flight Deck XM	XS	10	Google z przycimnianą szybką	https://www.gotravels.pl/images/Google/G_101/Gogle_narciarskie_Arctica_G-101C.jpg	Gogle	false	Usuń	Aktualizuj
Kijki narciarskie Leki Worldcup Racing SL TBS	XL	15	Lekkie solidne kijki dla zaawansowanych narciarzy	https://www.projektjunior.pl/21071-large_default/kije-narciarskie-dla-dzieci-rossignol-telescopic-jr-.jpg	Kijki	false	Usuń	Aktualizuj

Prev

Page 1 of 3

Next

Dodaj nowy sprzęt

Nazwa

Rozmiar

Cena

Opis

Zdjęcie

Kategoria

Dodaj sprzęt

Błędne dane

Dodaj nowy sprzęt

Nazwa

nazwa

Rozmiar

Rozmiar nie może być pusty

Cena

-7

Cena musi być liczbą większą od 0.

Opis

Zdjęcie

Proszę podać link do zdjęcia.

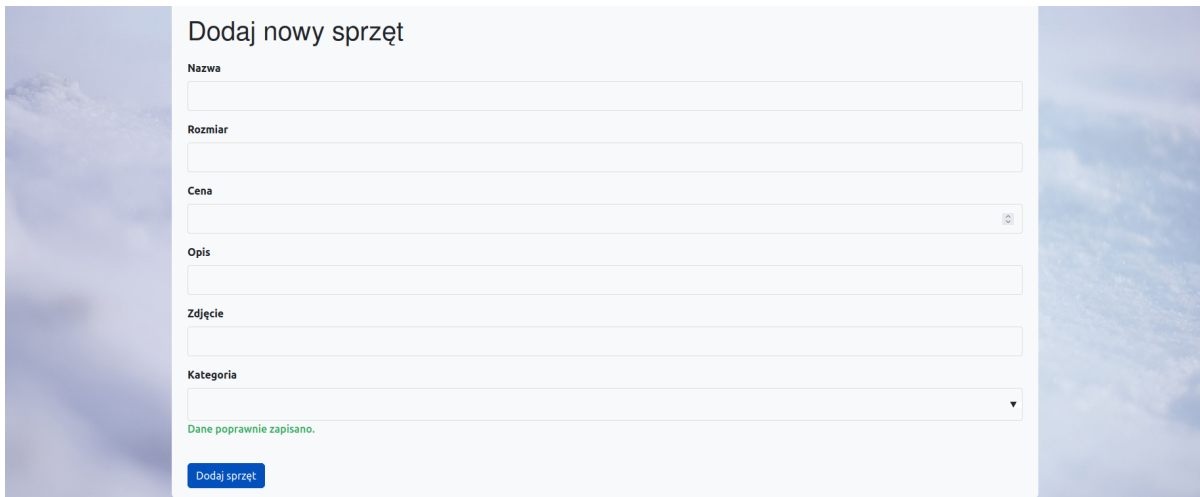
Kategoria

Proszę wybrać kategorię.

Proszę wypełnić wskazane pola formularza.

Dodaj sprzęt

Po wpisaniu poprawnych danych



Dodaj nowy sprzęt

Nazwa

Rozmiar

Cena

Opis

Zdjęcie

Kategoria

Dane poprawnie zapisano.

Dodaj sprzęt

## Sekcja dotycząca wypożyczeń

W tej sekcji dane dodane w poprzednich tabelach są wyświetlane w sposób przyjazny dla użytkownika. Jest tutaj zaimplementowana prosta funkcjonalność, która pozwala wypożyczyć wybrany sprzęt. Dodatkowo można filtrować sprzęty po kategorii. Aplikacja zlicza również, ile sprzętów aktualnie jest wypożyczonych.

Metody potrzebne do wysyłania odpowiednich żądań do serwera prezentują się następująco:

```
import CustomNavbar from "@/components/utils/CustomNavbar.vue";
import EquipmentForm from "@/components/equipment/EquipmentForm.vue";
import { apiLocation } from "@/components/utils/Properties";

export default {
  name: 'Borrowing',
  components: { CustomNavbar, EquipmentForm },

  data() {
    return {
      equipments: [],
      categories: [],
      amountOfBorrowedEquipment: '',
      currentCategory: 'Wszystkie',
      serverAvailable: true,
    }
  },

  mounted() {
    this.getCategories()
    this.getEquipments()
```

```

    this.getAmountOfBorrowedEquipment()
  },
  methods: {
    async getEquipments() {
      try {
        const response = await fetch(`${apiLocation}/get/equipments`)
        let equipments = await response.json()
        this.equipments = equipments
        this.equipments.sort((a, b) => (a.id > b.id) ? 1 : -1)
        this.serverAvailable = true
      } catch (error) {
        console.error(error)
        this.serverAvailable = false
      }
    },
    async getCategories() {
      try {
        const response = await fetch(`${apiLocation}/get/categories`)
        this.categories = await response.json()
        this.categories.sort((a, b) => (a.id > b.id) ? 1 : -1)
        this.serverAvailable = true
      } catch (error) {
        console.error(error)
        this.serverAvailable = false
      }
    },
    async getEquipmentsByCategory() {
      try {
        const currentCategory = this.categories.find(c => c.name ===
this.currentCategory)
        const response = await
fetch(`${apiLocation}/get/equipments/category/${currentCategory.id}`)
        this.equipments = await response.json()
        this.equipments.sort((a, b) => (a.id > b.id) ? 1 : -1)
        this.serverAvailable = true
      } catch (error) {
        console.error(error)
        this.serverAvailable = false
      }
    },
    async updateView() {
      if (this.currentCategory === 'Wszystkie') {
        await this.getEquipments()
        await this.getAmountOfBorrowedEquipment()
      }
    }
  }
}

```

```

    } else {
      await this.getEquipmentsByCategory()
      await this.getAmountOfBorrowedEquipmentByCategory()
    }
  },
  async borrowEquipment(id) {
    try {
      await fetch(`${apiLocation}/change-borrow/equipment/${id}`)
      await this.updateView()
    } catch (error) {
      console.error(error)
    }
  },

  async getAmountOfBorrowedEquipment() {
    try {
      const response = await
fetch(`${apiLocation}/get/equipments/countBorrowed`)
      const data = await response.json()
      this.amountOfBorrowedEquipment = data
    } catch (error) {
      console.error(error)
    }
  },

  async getAmountOfBorrowedEquipmentByCategory() {
    try {
      const currentCategory = this.categories.find(c => c.name ===
this.currentCategory)
      const response = await
fetch(`${apiLocation}/get/equipments/countBorrowed/category/${currentCategory.
id}`)
      const data = await response.json()
      this.amountOfBorrowedEquipment = data

    } catch (error) {
      console.error(error)
    }
  },

  getAmountOfAvaliableEquipment() {
    return this.equipments.length - this.amountOfBorrowedEquipment
  },

  clearStatus() {
    this.success = false

```



```

    this.error = false
  },
},
}
</script>

```

Kod html

```

<template>
  <CustomNavbar></CustomNavbar>
  <div class="container bg-light pt-3 mt-5 mb-5 pb-5 rounded-3">
    <div class="container-fluid mt-3">
      <h4 v-if="!this.serverAvailable" class="error-message">
        Nie udało się nawiązać połączenia z serwerem
      </h4>
      <h3>Filtruj</h3>
      <select v-model="currentCategory" @change="updateView()">
        <option @focus="clearStatus">Wszystkie</option>
        <option
          v-for="c in categories" :key="c.id"
          @focus="clearStatus"
        >
          {{ c.name }}
        </option>
      </select>
      <h2>Liczba aktualnie dostępnych sprzętów w kategorii:
        {{ this.getAmountOfAvaliableEquipment() }} / {{
this.equipments.length}}</h2>
      <div class="row mt-3">
        <div v-for="item in this.equipments" :key="item.id" class="col-12
col-sm-6 col-md-4 col-lg-3 mb-5">
          <div class="card h-100">
            <div>
              <p style="color: #32a95d" v-if="!item.borrowed">Dostępny</p>
              <p style="color: #d33c40" v-if="item.borrowed">Niedostępny</p>
            </div>
            <div class="h-75 d-flex align-items-center justify-content-center">
              
            </div>
            <div class="card-body text-center">
              {{ item.name }}
              <br>
              Rozmiar: {{ item.size }}
              <br>
              Cena: {{ item.price }} zł
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>

```

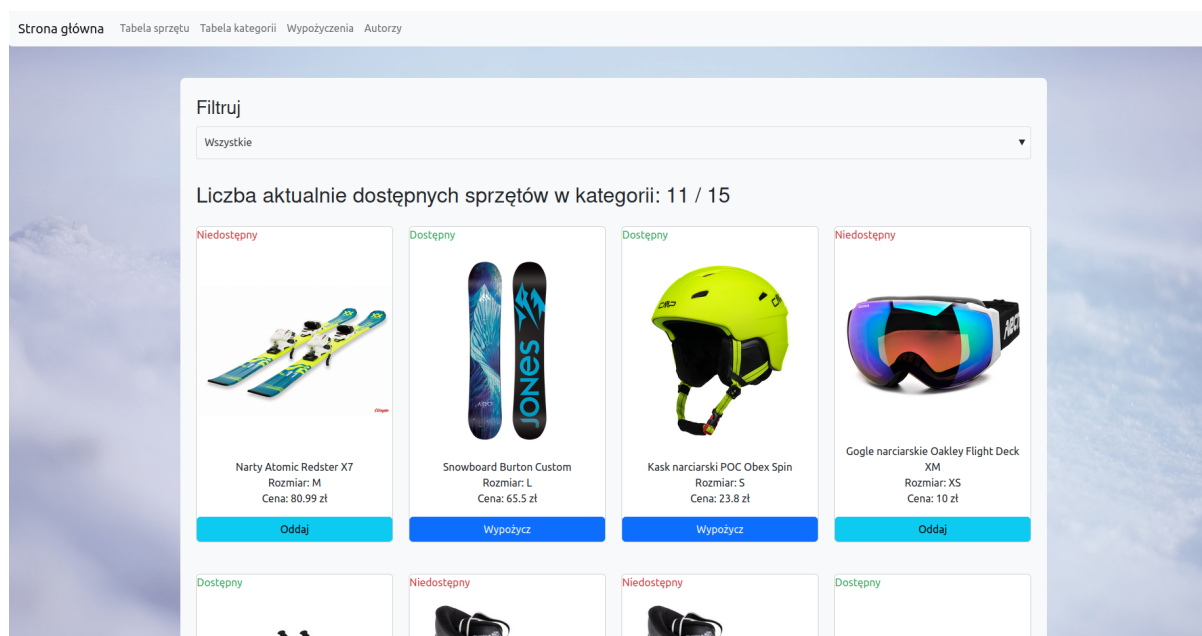
```

        <button v-if="!item.borrowed" class="btn btn-primary"
@click="borrowEquipment(item.id)">Wypożycz</button>
        <button v-if="item.borrowed" class="btn btn-info"
@click="borrowEquipment(item.id)">Oddaj</button>
    </div>
</div>
</div>
</div>
</div>
</template>

<script>

```

## Wygląd



## Filtrowanie po kategorii

Filtruj

Buty narciarskie

Liczba aktualnie dostępnych sprzętów w kategorii: 1 / 2

Niedostępny



Buty narciarskie Atomic Hawx Prime  
120 S  
Rozmiar: M  
Cena: 30 zł

Oddaj

Dostępny



Buty narciarskie Atomic Hawx Prime  
110 S  
Rozmiar: S  
Cena: 45 zł

Wypożycz

## Sekcja dotycząca autorów projektu

W projekcie znajduje się również zakładka wyświetlająca dane dotyczące autorów projektu oraz maszyny, na jakiej aplikacja została uruchomiona. Dane są ładowane z serwera.

```
<script>
```

```
import {apiLocation} from "@/components/utils/Properties";  
import CustomNavbar from "@/components/utils/CustomNavbar.vue";
```

```
export default {  
  name: 'Authors',  
  components: {CustomNavbar},
```

```
  data() {  
    return {  
      authors: {  
        author1: "",  
        author2: "",  
        datetime: "",  
        javaVersion: "",  
        userName: "",  
        osName: "",  
        localAddress: ""  
      },  
      serverAvailable: true,  
    }  
  },  
}
```

```

mounted() {
  this.getAuthors()
},
methods: {
  async getAuthors() {
    try {
      const response = await fetch(`${apiLocation}/get/authors`)
      let authors = await response.json()
      this.authors = authors
      this.serverAvailable = true
    } catch (error) {
      console.error(error)
      this.serverAvailable = false
    }
  },
},
}
</script>

```

```

<template>
  <CustomNavbar></CustomNavbar>
  <div class="container bg-light pt-3 mt-5 mb-5 pb-5 rounded-3">
    <div class="container-fluid mt-3">
      <h4 v-if="!this.serverAvailable" class="error-message">
        Nie udało się nawiązać połączenia z serwerem
      </h4>
      <h1>Autorzy / Informacje</h1>

      <h3> Autorzy: {{ authors.author1 }}, {{ authors.author2 }} </h3>
      <h3> Data i czas: {{ authors.datetime }} </h3>
      <h3> Wersja Javy: {{ authors.javaVersion }} </h3>
      <h3> Nazwa użytkownika: {{ authors.userName }} </h3>
      <h3> Nazwa systemu operacyjnego: {{ authors.osName }} </h3>
      <h3> Adres IP: {{ authors.localAddress }} </h3>

      <button class="btn btn-primary "
        @click="this.$router.push({ name: 'main'})">Powrót
      </button>
    </div>
  </div>
</template>

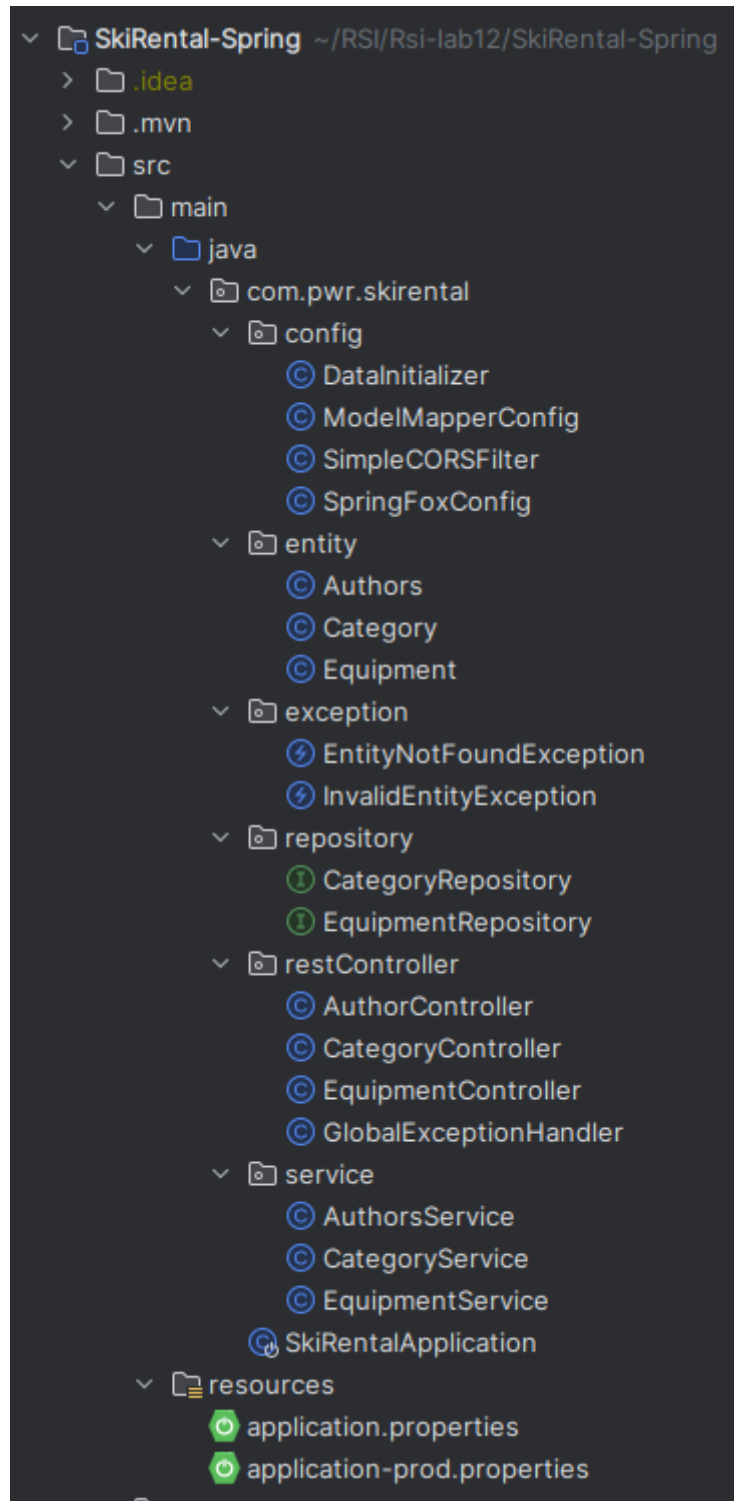
```

## Główne kroki projektowania backendu

Projekt składa się z 5 części:

- Konfiguracji
- Encji
- Warstwy kontrolerów
- Warstwy serwisów
- Warstwy repozytoriów

Cała struktura projektu prezentuje się następująco:



Mamy tutaj 3 encje, dla każdej z nich struktura kodu przebiega dosyć analogicznie więc zostanie omówiona tylko jedna z nich - Equipment

## Encja Equipment

Zdefiniowano pola, podstawową walidację, getters, setters, konstruktory

```
package com.pwr.skirental.entity;

import lombok.*;
import org.hibernate.Hibernate;

import javax.persistence.*;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import java.util.Objects;

@Getter
@Setter
@ToString
@RequiredArgsConstructor
@Builder
@AllArgsConstructor
@Entity
public class Equipment {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```

private Long id;

@NotBlank(message = "Name is mandatory")
private String name;

@ManyToOne
@JoinColumn

@NotNull(message = "Category is mandatory")
private Category category;

private String description;

@NotBlank(message = "Size is mandatory")
private String size;

@Min(value = 0, message = "Price must be greater than 0")
private double price;

@NotBlank(message = "Image is mandatory")
private String image;

private boolean isBorrowed;

@Override

public boolean equals(Object o) {

    if (this == o) return true;

    if (o == null || Hibernate.getClass(this) !=
Hibernate.getClass(o)) return false;

    Equipment equipment = (Equipment) o;

    return getId() != null && Objects.equals(getId(),
equipment.getId());

}

@Override

public int hashCode() {

    return getClass().hashCode();

}

```

```
}
```

## Kontroler

Zostały zdefiniowane wszystkie wykorzystywane metody CRUD. Logiką działania tych metod zajmuje się serwis, który jest w kolejnej warstwie.

```
@RestController
@RequiredArgsConstructor
public class EquipmentController {

    private final EquipmentService equipmentService;

    @GetMapping("/")
    public String home() {
        return "Home";
    }

    @GetMapping("/get/equipments")
    public ResponseEntity<Collection<Equipment>> getEquipments() {
        return ResponseEntity.ok(equipmentService.getEquipments());
    }

    @GetMapping("/get/equipment/{id}")
    public ResponseEntity<Equipment> getEquipment(@PathVariable Long id) {
        try {
            return ResponseEntity.ok(equipmentService.getEquipment(id));
        } catch (EntityNotFoundException e) {
            return ResponseEntity.notFound().build();
        }
    }
}
```



```

    }

    @PostMapping("/post/equipment")

    public ResponseEntity<?> addEquipment(@Valid @RequestBody Equipment
equipment) {

        try {

            return
ResponseEntity.ok(equipmentService.addEquipment(equipment));

        } catch (InvalidEntityException e) {

            return ResponseEntity.badRequest().body("Invalid entity");

        }

    }

    @PatchMapping("/patch/equipment")

    public ResponseEntity<?> updateEquipment(@Valid @RequestBody Equipment
equipment) {

        try {

            return
ResponseEntity.ok(equipmentService.updateEquipment(equipment));

        } catch (EntityNotFoundException e) {

            return ResponseEntity.badRequest().body("Entity not found");

        } catch (InvalidEntityException e) {

            return ResponseEntity.badRequest().body("Invalid entity");

        }

    }

    @DeleteMapping("/delete/equipment/{id}")

    public ResponseEntity<?> deleteEquipment(@PathVariable Long id) {

        try {

            equipmentService.deleteEquipment(id);

            return ResponseEntity.ok().build();

        } catch (EntityNotFoundException e) {

```

```

        return ResponseEntity.notFound().build();
    }
}

@GetMapping("/change-borrow/equipment/{id}")
public ResponseEntity<?> changeBorrowStatus(@PathVariable Long id) {
    try {
        equipmentService.changeBorrowStatus(id);
        return ResponseEntity.ok().build();
    } catch (EntityNotFoundException e) {
        return ResponseEntity.notFound().build();
    }
}

@GetMapping("/get/equipments/category/{categoryId}")
public ResponseEntity<List<Equipment>>
getEquipmentsByCategory(@PathVariable Long categoryId) {
    try {
        return
ResponseEntity.ok(equipmentService.getEquipmentsByCategory(categoryId));
    } catch (EntityNotFoundException e) {
        return ResponseEntity.notFound().build();
    }
}

@GetMapping("/get/equipments/size/{size}")
public ResponseEntity<List<Equipment>>
getEquipmentsBySize(@PathVariable String size) {
    return
ResponseEntity.ok(equipmentService.getEquipmentsBySize(size));
}

@GetMapping("/get/equipments/countBorrowed")

```

```

public ResponseEntity<Integer> getEquipmentsCountBorrowed() {
    return ResponseEntity.ok(equipmentService.countBorrowed());
}

@GetMapping("/get/equipments/countBorrowed/category/{categoryId}")
public ResponseEntity<Integer>
getEquipmentsCountBorrowedByCategory(@PathVariable Long categoryId) {
    return
ResponseEntity.ok(equipmentService.countBorrowed(categoryId));
}
}

```

## Serwis

Zajmuje się on przetworzeniem danych przed zapisem do bazy. Przy pomocy repozytorium wykonuje operacje na bazie danych.

```

@Service
@RequiredArgsConstructor
public class EquipmentService {

    private final EquipmentRepository equipmentRepository;
    private final ModelMapper modelMapper;

    public Collection<Equipment> getEquipments() {
        return equipmentRepository.findAll();
    }

    public Equipment getEquipment(Long id) throws EntityNotFoundException
{

```

```

        return
equipmentRepository.findById(id).orElseThrow(EntityNotFoundException::new
);

    }

    public Equipment addEquipment(Equipment equipment) throws
InvalidEntityException {

        return equipmentRepository.save(equipment);

    }

    public Equipment updateEquipment(Equipment equipment) throws
EntityNotFoundException, InvalidEntityException {

        Optional<Equipment> equipmentDb =
equipmentRepository.findById(equipment.getId());

        if (equipmentDb.isEmpty()) {

            throw new EntityNotFoundException();

        }

        return equipmentRepository.save(equipment);

    }

    public void deleteEquipment(Long id) throws EntityNotFoundException {

        Optional<Equipment> equipment = equipmentRepository.findById(id);

        if (equipment.isEmpty()) {

            throw new EntityNotFoundException();

        }

        equipmentRepository.deleteById(id);

    }

    public void changeBorrowStatus(Long id) throws EntityNotFoundException
{

        Optional<Equipment> equipment = equipmentRepository.findById(id);

        if (equipment.isEmpty()) {

            throw new EntityNotFoundException();

        }

    }

```

```

        equipment.get().setBorrowed(!equipment.get().isBorrowed());

        equipmentRepository.save(equipment.get());
    }

    public List<Equipment> getEquipmentsByCategory(Long categoryId) throws
EntityNotFoundException {

        return equipmentRepository.findByCategoryId(categoryId);
    }

    public Integer countBorrowed() {

        return equipmentRepository.countByIsBorrowed(true);
    }

    public Integer countBorrowed(Long categoryId) {

        return equipmentRepository.countByIsBorrowedAndCategoryId(true,
categoryId);
    }

    public List<Equipment> getEquipmentsBySize(String size) {

        return equipmentRepository.findBySize(size);}

```

## Repozytorium

Wykonuje operacje na bazie. Wystarczy zdefiniować interfejs, metody implementują się automatycznie.

```

public interface EquipmentRepository extends JpaRepository<Equipment,
Long> {

    List<Equipment> findByCategoryId(Long name);

    Integer countByIsBorrowed(boolean borrowed);

    Integer countByIsBorrowedAndCategoryId(boolean borrowed, Long
categoryId);

    List<Equipment> findBySize(String size);

}

```

# Swagger

Całe api zostało udokumentowane przy pomocy Swaggera. Do projektu został on dodany przy pomocy dependency io.springfox

<b>author-controller</b> Author Controller		▼
GET	/get/authors	getAuthors
<b>basic-error-controller</b> Basic Error Controller		>
<b>category-controller</b> Category Controller		▼
DELETE	/delete/category/{id}	deleteAuthor
GET	/get/categories	getAuthors
GET	/get/category/{id}	getAuthor
PATCH	/patch/category	updateAuthor
POST	/post/category	addAuthor
<b>equipment-controller</b> Equipment Controller		▼
GET	/	home
GET	/change-borrow/equipment/{id}	changeBorrowStatus
DELETE	/delete/equipment/{id}	deleteEquipment
GET	/get/equipment/{id}	getEquipment
GET	/get/equipments	getEquipments
GET	/get/equipments/category/{categoryId}	getEquipmentsByCategory

## Wdrożenie na chmurę

Cała aplikacja została zdokeryzowana. Dzięki temu wgranie jej na chmurę było o wiele łatwiejsze. Zostały przygotowane odpowiednie pliki DockerFile, następnie utworzone obrazy Fronendu, Backendu oraz Bazy danych i uruchomione na chmurze przy pomocy komendy docker-compose up

docker-compose.yml projektu prezentuje się następująco:

```
version: '3'

services:
  db:
    container_name: db
    image: postgres:15-alpine
```

```
restart: always

environment:
  - POSTGRES_USER=postgres
  - POSTGRES_PASSWORD=postgres
  - POSTGRES_DB=ski_rental

ports:
  - "5432:5432"

volumes:
  - db:/var/lib/postgresql/data

backend:

  container_name: backend

  build:
    context: ./SkiRental-Spring
    dockerfile: Dockerfile

  ports:
    - "8080:8080"

  restart: always

  depends_on:
    - db

  environment:
    - SPRING_PROFILES_ACTIVE=prod
    - POSTGRES_URL=jdbc:postgresql://db:5432/ski_rental
    - POSTGRES_USER=postgres
    - POSTGRES_PASSWORD=postgres

frontend:
```

```
container_name: frontend

build:

  context: ./SkiRental-Vue/

  dockerfile: Dockerfile

ports:

  - "80:80"

restart: always

depends_on:

  - backend

environment:

  - NODE_ENV=production

  - API_URL=http://localhost:8080

expose:

  - "80"

volumes:

  db:

    driver: local
```

Dodatkowo trzeba było obsłużyć mechanizm CORS. W tym celu na backendzie został uruchomiony odpowiedni filtr, który wpuszcza requesty tylko z ip serwera, na którym postawiona jest aplikacja.

```
@Override

public void doFilter(ServletRequest req, ServletResponse res, FilterChain
chain) throws IOException, ServletException {

    HttpServletRequest request = (HttpServletRequest) req;

    HttpServletResponse response = (HttpServletResponse) res;
```



```
response.setHeader("Access-Control-Allow-Origin",  
"http://16.16.160.55");  
  
response.setHeader("Access-Control-Allow-Credentials", "true");  
  
response.setHeader("Access-Control-Allow-Methods", "POST, GET, PATCH,  
OPTIONS, DELETE, PUT");  
  
response.setHeader("Access-Control-Max-Age", "3600");  
  
response.setHeader("Access-Control-Allow-Headers", "Content-Type,  
Accept, X-Requested-With, remember-me");  
  
chain.doFilter(req, res);  
}
```

Aplikacja została wdrożona na chmurę AWS. Została tam stworzona maszyna linux przy pomocy usługi EC2. Do serwera można połączyć się przy pomocy usługi ssh oraz specjalnie wygenerowanego klucza prywatnego. Na maszynie został zainstalowany docker według oficjalnej dokumentacji. Następnie projekt został sklonowany z repozytorium GitHub oraz uruchomiony przy pomocy komendy docker compose up.

```
ubuntu@ip-172-31-8-225:~/RSI/Rsi-lab12$ sudo docker compose up  
[+] Building 0.0s (0/0)  
[+] Running 4/3  
✓ Network rsi-lab12_default Created  
✓ Container db Created  
✓ Container backend Created  
✓ Container frontend Created
```

Aplikację można przeglądać pod adresem <http://16.16.160.55>

## Modyfikacja

Podczas zajęć została przeprowadzona drobna modyfikacja stanu projektu.

Zostały dodane nowe narty - Narty, nowa kategoria - sporty wodne oraz nowy sprzęt - deska surfingowa. Narty zostały zaktualizowane do nazwy Narty2.


Dane po modyfikacji prezentują się następująco

Nazwa	Rozmiar	Cena	Opis	Zdjęcie	Kategoria	Czy wypożyczona	Usuń	Aktualizuj
Snowboard Burton Custom X	L	90	Snowboard dla zaawansowanych snowboardzistów	https://snowboardok.pl/21149-large_default/p1530-snowboard-jones-w-airheart.jpg	Snowboard	false	Usuń	Aktualizuj
Narty Atomic Redster X9 S FT	XL	200	Narty slalomowe dla zaawansowanych narciarzy	https://olimpiasport.pl/upload/79/olimpiadb_79748_d.jpg	Narty	false	Usuń	Aktualizuj
Narty2	XXL	1000	Opis2	https://www.nartywarszawa.pl/25069-large_default/narty-stockli-laser-wrt-srt12-carbon-2024.jpg	Narty	false	Usuń	Aktualizuj
Deska surfingowa	1,90	3000	To opis sufringu	https://www.caramella.pl/cache/files/990148018/deskisurfingowe3---w-1200.jpg	sporty wodne	false	Usuń	Aktualizuj

Nazwa	Usuń	Aktualizuj
Kijki	Usuń	Aktualizuj
sporty wodne	Usuń	Aktualizuj


[Prev](#) Page 3 of 3 [Next](#)

Nowe sprzęty możemy podziwiać oraz wypożyczać w sekcji wypożyczeń




Kask narciarski POC Obex Pure  
Rozmiar: L  
Cena: 15.3 zł

Wypożycz




Snowboard Burton Custom Flying V  
Rozmiar: M  
Cena: 70 zł

Wypożycz



Snowboard Burton Custom X  
Rozmiar: L  
Cena: 90 zł


Wypożycz



Narty Atomic Redster X9 S FT  
Rozmiar: XL  
Cena: 200 zł

Wypożycz


Dostępny



Narty2  
Rozmiar: XXL  
Cena: 1000 zł

Wypożycz

Dostępny



Deska surfingowa  
Rozmiar: 1,90  
Cena: 3000 zł

Wypożycz