Politechnika Warszawska
Ośrodek Kształcenia na Odległość
Wydział EiTI

Dokumentacja – projekt nr 2

Zaawansowane aplikacje internetowe

Dominik Kanthak
Studia magisterskie
Kierunek Informatyka
Indeks: 309478

1. Treść zadania projektu nr 1 (Javascript)

Opis funkcjonalny:

Stwórz prostą aplikację typu SPA (ang. Single Page Application) wykorzystującą wybrany framework front-endowy języka JavaScript. Aplikacja powinna udostępniać możliwość zarządzania wydarzeniami oznaczanymi na osi czasu (analogicznie jak w projekcie 1, ale wszystkie operacje mogą dziać się wyłącznie we frontendzie, bez komunikacji z bazą danych).

Cechy aplikacji:

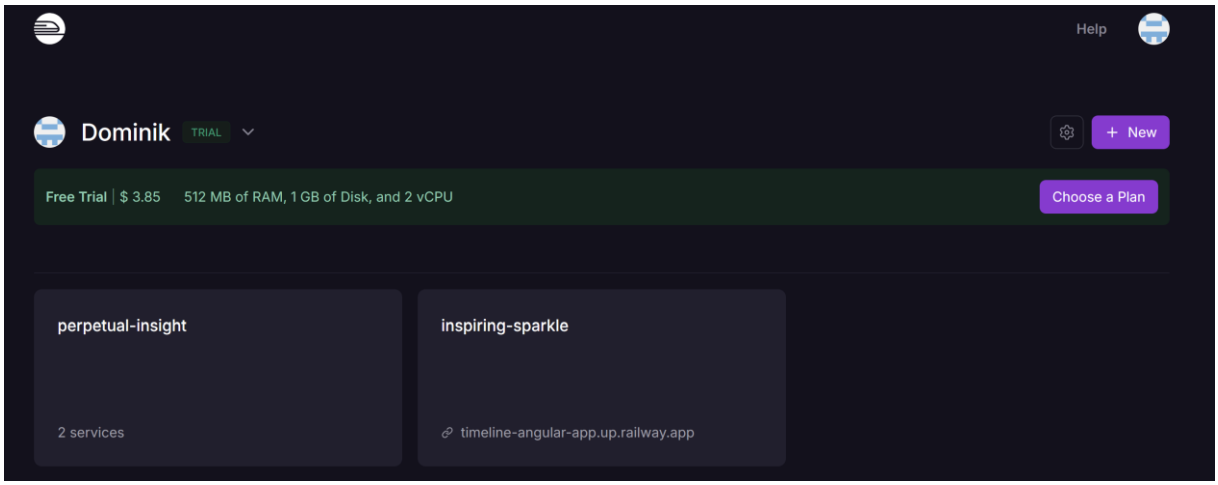
- możliwość dodawania, edycji oraz usuwania wydarzeń,
- aplikacja działa wyłącznie po stronie przeglądarki (bez części serwerowej), co oznacza, że informacje o zdarzeniach/procesach przechowywane są jedynie w pamięci przeglądarki,
- po uruchomieniu aplikacji na osi czasu znajdują się przykładowe wydarzenia,
- wydarzenia powinny być prezentowane w estetycznej formie graficznej (można skorzystać z wybranej przez siebie biblioteki JavaScript) oraz w tabeli,
- cechy wydarzenia takie same jak w projekcie 1,
- tabela z wydarzeniami może zostać posortowana wedle wybranej cechy (nazwa, data, itp.),
- tabela wydarzeń może być filtrowana po dacie przez wpisanie ograniczeń „od” i/lub „do”.

Specyfikacja techniczna (technologie, które należy użyć):

- wybrany przez Studenta framework front-endowy języka JavaScript, np. React, Vue, Angular,
- wybrany przez Studenta framework dla layoutu, np. Bootstrap, Semantic UI, itp.
- wybrana przez Studenta biblioteka JS wspomagająca rysowanie osi czasu,
- Node.js z menedżerem pakietów dla lokalnego zahostowania aplikacji.

2. Opis konfiguracji serwera i uruchomienia witryny

Aplikacja została uruchomiona na platformie Railway.app. Jest to platforma hostingowa typu PaaS (Platform as a Service), która umożliwia łatwe wdrażanie, zarządzanie i skalowanie aplikacji internetowych oraz baz danych bez konieczności posiadania zaawansowanej infrastruktury serwerowej. Railway oferuje wsparcie dla różnych języków programowania, takich jak Node.js, Python, Go, czy PHP, a także frameworków takich jak Angular, co ułatwia wdrożenie aplikacji na tej platformie.



Podgląd platformy Railway oraz uruchomione na niej aplikacje

Platforma ta została wybrana ze względu na mało skomplikowaną ciągłą procedurę wdrożenia aplikacji oraz darmową cenę. Oferuje ona konto TRAIL, które po poprawnym zarejestrowaniu i połączeniu z kontem Github (warunek konieczny do odsiania kont spamerskich) pozwala na wykorzystanie 5$ środków. Udostępnia ona wtedy zasoby, które pokazane zostały powyżej. Jednocześnie nie wymagana przy tym podpinania kart płatniczych. Dla czasowych jak i zasobowych wymagań projektu platforma ta wydaje się wystarczającym rozwiązaniem. W celu dodatkowego zaoszczędzenia środków – platforma pozwala uśpić udostępnione serwisy.

2.1. Uruchomienie lokalne

1. Pobranie oraz zainstalowanie Nodej.js oraz npm ze strony: https://nodejs.org/en
2. Należy pobrać i zainstalować Angular CLI, najlepiej poprzez wiersz poleceń/konsolę (komenda dla git bash zainstalowanego w systemie Windows):

   npm install -g @angular/cli

3. W kolejnym korku należy pobrać kod źródłowy z repozytorium Github:

   git clone https://github.com/kanthakdominik/timeline_angular.git
4. Przejść do aplikacji:

   cd timeline

5. Zainstaluj zależności aplikacji:

   npm install
6. Uruchom aplikację:

   npm run dev

Aplikacja lokalnie działa pod adresem:

http://localhost:4200

Alternatywnie aplikację można uruchomić także komendą uruchamiającą prosty serwer:

npm install

Aplikacja dostępna wtedy będzie pod adresem:

http://localhost:8080

Taka konfiguracja uruchamiania została użyta w celu udostępnienia konfiguracji dla aplikacji działającej na platformie Railway (CI/CD) oraz możliwości uruchomienia lokalnego.

2.2. Konfiguracja serwera

1. Utworzenie nowego konta platformie Railway, najlepiej powiązać je z kontem Github.

2. Utworzyć nowy projekt na platformie Railway.

3. Stworzenie serwisu aplikacji – poprzez wybranie określonego repozytorium z podpiętego konta Github. Po tym kroku Railway sam wykryje, zbuduje i wdroży aplikację (kontener z aplikacją). W projekcie używana jest tylko część frontendowa, stąd nie ma potrzeby tworzenia bazy danych.

4. Konfiguracja aplikacji – należy dodać odpowiedni kod, który będzie współpracował z platformą Railway:

   - do repozytorium został dodany plik server.js w głównym katalogu aplikacji, który wykorzystuje framework express.js. Służy do wystawienia prostego serwera, na którym będzie działać aplikacja produkcyjnie.

   - w pliku konfiguracyjnym package.json: dodać/zamienić instrukcje:

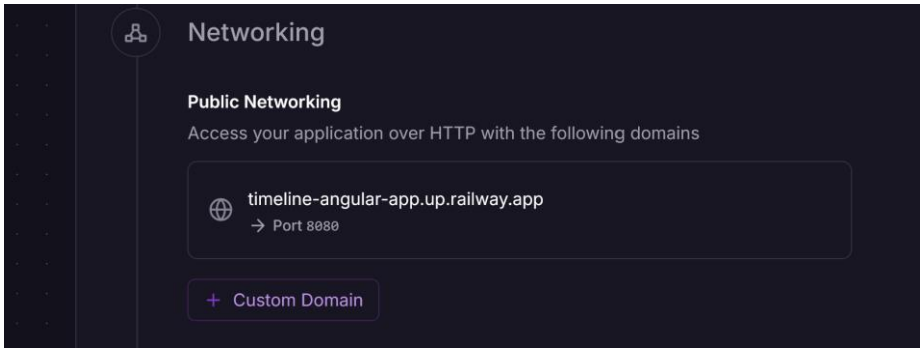
   "start": "node server.js",

   "build": "ng build --configuration production",

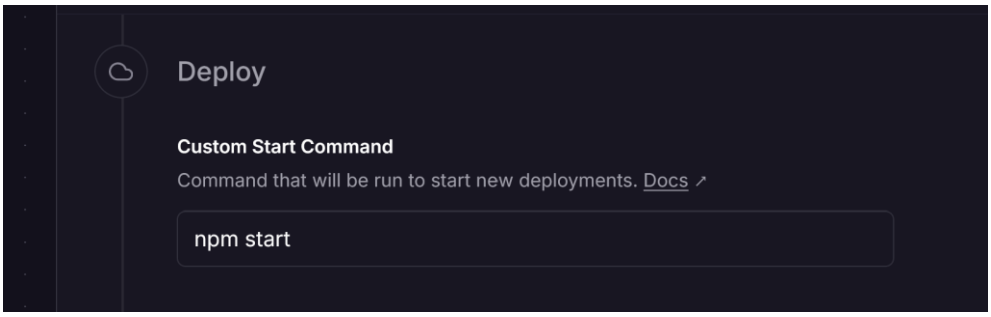
   oraz dodać nowy import zależności do express.js:
   "express": "^4.21.1",

   - w pliku .gitignore zakomentować/usunąć ignorowanie folderu ‘/dist’ – na serwerze produkcyjnym to właśnie z plików znajdujących się w nim jest udostępniana aplikacja.

5. Konfiguracja Railway – należy ustawić parametry projektu aplikacji, wchodząc na zakładkę Settings:



Wybranie domeny strony projektu oraz wybranie numeru portu, na którym jest udostępniana aplikacja



Zmiana/ustawienie komendy do wdrożenia aplikacji

6. Ewentualna konfiguracja innych ustawień – jak automatyczną budowę i wdrażanie po każdej zmianie wykrytej w repozytorium kodu, czy możliwość usypiania serwisów w momentach bezczynności.

2.3. Dane aplikacji

Aplikacja działa pod adresem:

https://timeline-angular-app.up.railway.app/

W ramach testowych oraz pokazania treści aplikacja jest uzupełniona danymi przedstawiającymi przykładowe wydarzenia oraz kategorie. Są także stworzone 2 konta administratorów:

Email	Hasło
admin@timeline.com	Password123!
admin2@timeline.com	Password123!

Repozytorium kodu:

https://github.com/kanthakdominik/timeline_angular

3. Opisu aplikacji i decyzje projektowe

Aplikacja została napisana w języku **Typescript / Javascript**. Bazuje ona na frameworku **Angular** - do tworzenia aplikacji webowych, stworzonym i utrzymywanym przez Google. Wykorzystuje on język typescript do silnego typowania zmiennych. Jest także modularny – stworzone widoki składają się z współpracujących ze sobą modułów. Posiada ona także wewnętrzny routing – pozwalający na łatwą nawigację pomiędzy komponentami w aplikacji typu SPA (single page).

Chociaż aplikacje napisane z pomocą Angulara to głównie aplikacje frontendowe, komunikujące się poprzez REST API do serwisów backendowych - nic nie stoi na przeszkodzie aby stworzyć tylko aplikację frontendową, bez konieczności użycia backendu. W takim przypadku wykorzystać można wykorzystać różne sposoby przechowywania danych przez przeglądarkę. Można wyróżnić sposoby:

- **In-Memory Storage** – dane są przechowywane w zmiennych aplikacji Angular, odświeżenie strony powoduje usunięcie danych.
- **Session Storage** - dane są przechowywane tylko na czas trwania sesji przeglądarki, zamknięcie karty lub przeglądarki powoduje usunięcie danych
- **Local Storage** - dane są przechowywane w przeglądarce i nie mają daty wygaśnięcia. Dane są dostępne nawet po zamknięciu i ponownym otwarciu przeglądarki.

W projekcie zdecydowano się na użycie **Session Storage** jako sposobu na przechowywanie danych. Pozwala to z jednej strony na prezentację wymaganych danych oraz operacje na nich w obrębie jednej sesji, z drugiej strony pozwala także zachować elastyczność – projekt miał zasadniczo być frontendowy – by zresetować stan aplikacji wystarczy wyłączyć kartę/przeglądarkę i ponownie wejść na stronę.

W projekcie wykorzystano również bibliotekę **Bootstrap**. Pozwala on na wykorzystanie gotowych, wystylizowanych komponentów (wykorzystujących HTML, CSS, Javascript). Z jego pomocą tworzony jest też wizualny komponent linii czasu w projekcie.

Pomocną okazała się także biblioteka **ng-bootstrap.** Jest to biblioteka, która integruje komponenty Bootstrap z Angular, umożliwiając ich użycie bez konieczności stosowania jQuery. Biblioteka ta jest zaprojektowana tak, aby działać bezproblemowo z Angular, oferując różne komponenty, takie jak modale, tooltipy i inne.

Oprócz wyżej wymienionych technologii, w projekcie wykorzystano także:

- Bibliotekę **crypto-js -** używaną do hash'owania haseł i porównywania hash'y haseł użytkowników.
- Bibliotekę **rxjs** - używaną do operacji asynchronicznych
- **Express.js** – do hostowania aplikacji

4. Opisu kodu stworzonej aplikacji

Aplikacja składa się z komponentów - każdy z nich zawiera zawsze trzy pliki:

- plik .html - odpowiedzialny za układ widoku komponentu,
- plik .css – odpowiedzialny za dedykowane style wyświetlania komponentu
- plik .ts – odpowiedzialny za logikę komponentu, jego stan oraz zachowanie

Komponenty są podzielone na foldery grupujące je o podobnej funkcjonalności:

- Folder **auth** – znajdują się tutaj komponenty:
  - **register** – komponent strony z formularzem rejestracji do aplikacji
  - **login** – komponent strony z formularzem logowania do aplikacji
  - **change-password** – komponent strony z formularzem zmiany hasła użytkownika

  Posiadają one także walidację danych wprowadzanych przez użytkownika oraz możliwość wyświetlenia błędów walidacji użytkownikowi.

```
this.registerForm = this.fb.group({
  name: ['', [
    Validators.required,
    Validators.maxLength(50)
  ]],
  email: ['', [
    Validators.required,
    Validators.email,
    Validators.maxLength(100)]],
  password: ['', [
    Validators.required,
    Validators.minLength(8),
    Validators.pattern(/^(?=.*[A-Z])(?=.*[a-z])(?=.*[0-9])(?=.*[!@#$%^&*()_+\-=\[\]{};':"\\|,.<>\/?])[A-Za-z\d!
  password_confirmation: ['', Validators.required]
}, {
  validators: TimelineValidators.passwordMatch
});
```

- Folder **bars** – znajdują się tutaj komponenty:

  - **categories-bar** – komponent widoku paska z wyborem kategorii. Zawiera także przycisk służący do podglądu wydruku strony oraz przycisk rozwijający / zwijający karty z wydarzeniami. Dodatkowo dla zalogowanego użytkownika wyświetla także przyciski służące do edycji koloru, edycji nazwy oraz usunięcia kategorii. Pozwala także na filtrowanie wydarzeń poprzez kategorie.

  - **filter-bar** - komponent widoku paska z widgetami odpowiedzialnymi za filtrowanie po datach oraz sortowanie, widoczny jedynie w trybie tabeli.

- **main-bar** - komponent widoku głównego paska strony. Zawiera logo strony, przyciski służące do rejestracji, logowania, zmiany hasła oraz wylogowania. Wyświetla także nazwę obecnie zalogowanego użytkownika.

- **settings-bar** - komponent widoku paska widocznego jedynie dla zalogowanego użytkownika. Wyświetla przyciski służące do dodawania nowego wydarzenia lub dodania nowej kategorii.
- **view-mode-bar -** komponent widoku paska pozwalającego na zmianę trybu wyświetlania aplikacji na tryb tabeli lub tryb osi czasu.

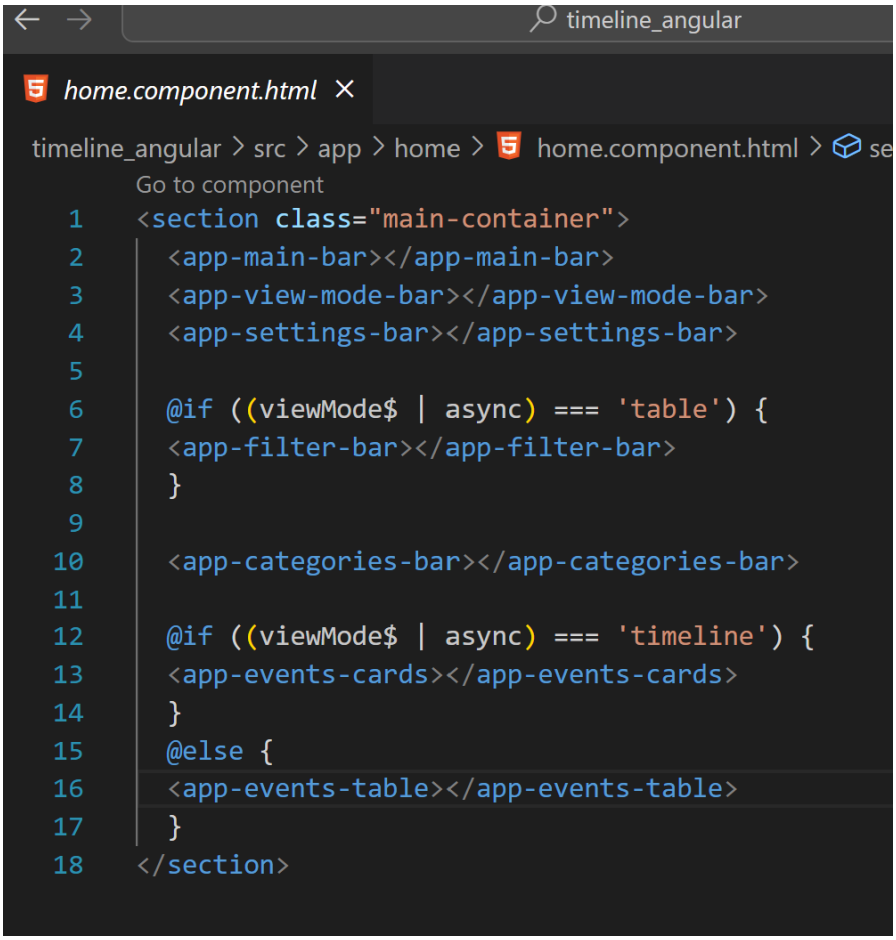
```
@Component({
  selector: 'app-categories-bar',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './categories-bar.component.html',
  styleUrl: './categories-bar.component.css'
})
export class CategoriesBarComponent implements OnInit, AfterViewChecked {
  categories: Category[] = [];
  areAllCardsExpanded: boolean = false;
  activeCategoryId: number | null = null;
  private shouldPrint = false;
  isLoggedIn$!: Observable<boolean>;
  isTimelineView: boolean = true;

  constructor(
    private modalService: NgbModal,
    private dataService: DataService,
    private authService: AuthService,
    private cdr: ChangeDetectorRef
  ) { }
```

- Folder **events** – znajdują się tutaj komponenty:

  - **events-cards** - komponent osi czasu wraz kartami, w których wyświetlane są w kolejności wydarzenia. Każda taka karta posiada także możliwość edycji czy usunięcia dla zalogowanego użytkownika.

  - **events-table** - komponent tabeli w której wyświetlane są w kolejności wydarzenia. Może być ona filtrowana po datach, kategoriach oraz sortowana alfabetycznie po nazwach, datach, kategoriach wydarzeń.

- Folder **home** – znajduje się tutaj komponent:
    - **home** - komponent grupujący powyższe komponenty oraz obsługujący logikę wyświetlania trybu kard lub trybu tabeli.



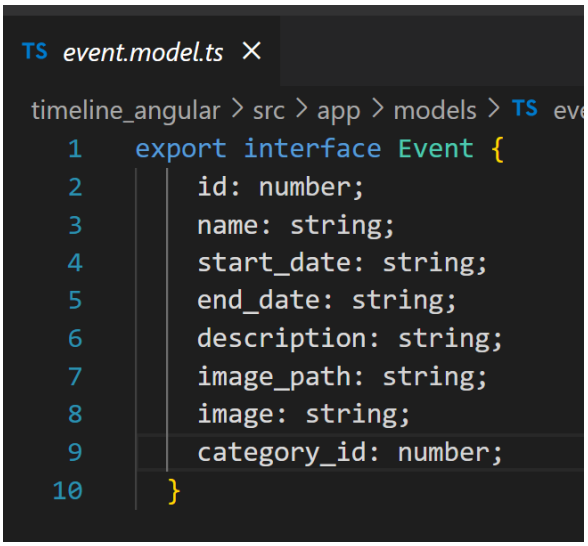
- Folder **modals** – znajdują się tutaj komponenty:
    - **add-category-modal** - modal pozwalający wprowadzić dane podczas tworzenia nowej kategorii.
    - **add-event-modal** - modal pozwalający wprowadzić dane podczas tworzenia nowego wydarzenia.
    - **edit-event-modal** - modal pozwalający wprowadzić dane podczas edycji wydarzenia.
    - **change-category-color-modal** - modal pozwalający wprowadzić dane podczas edycji koloru kategorii.

- **change-category-name-modal** - modal pozwalający wprowadzić dane podczas edycji nazwy kategorii.

- **confirmation-modal** - modal wyświetlające pytanie o potwierdzenie danej akcji wykonywanej przez użytkownika

```
@Component({
  selector: 'app-confirmation-modal',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './confirmation-modal.component.html'
})
export class ConfirmationModalComponent {
  @Input() title: string = '';
  @Input() message: string = '';
  @Input() confirmButtonText: string = 'Potwierdź';
  @Input() cancelButtonText: string = 'Anuluj';
  @Input() confirmButtonVisible: boolean = true;

  constructor(public modal: NgbActiveModal) {}
}
```

- Folder **models** – znajdują się tutaj interfejsy typescript'owe: **Category**, **Event**, **User**. Są to abstrakcje opisujące model danych aplikacji.



- Folder **services** – znajdują się tutaj pliki:

    - **auth.service.ts** – jest to serwis zarządzający stanem uwierzytelnienia użytkownika. Używa funkcjonalności BehaviorSubject, aby śledzić status logowania i zalogowanego użytkownika. Jest on inicjalizowany danymi przechowywanymi w Session Storage. Udostępnia on innym komponentom metody logowania, wylogowania oraz pobierania danych o bieżącym użytkowniku.

    - **data.service.ts** – jest to serwis odpowiedzialny za zarządzanie danymi w aplikacji. Inicjuje przykładowe dane wydarzeń ,kategorii oraz użytkowników. Posiada metody do składowania danych w Session Storage oraz udostępnia metody typu CRUD (Create, Read, Update, Delete) dla innych komponentów wykorzystujących dane. Serwis także wykorzystuje BehaviorSubject oraz używa biblioteki RxJs. Znajdują się w nim także dodatkowe metody do hash'owania haseł oraz do konwersji plików obrazów do formatu base64**.**

```
export class DataService {
  private categoriesSubject = new BehaviorSubject<Category[]>(this.loadCategoriesFromSessionStorage());
  private activeCategoryFilterSubject = new BehaviorSubject<number | null>(null);
  private eventsSubject = new BehaviorSubject<Event[]>([]);
  private cardsExpandedStateSubject = new BehaviorSubject<boolean>(false);
  private usersSubject = new BehaviorSubject<User[]>(this.loadUsersFromSessionStorage());
  private viewModeSubject = new BehaviorSubject<ViewMode>('timeline');
  private sortConfigSubject = new BehaviorSubject<SortConfig>({ field: 'name', direction: 'asc' });
  private dateFilterSubject = new BehaviorSubject<DateFilter>({ startDate: null, endDate: null });

  activeCategoryFilter$ = this.activeCategoryFilterSubject.asObservable();
  categories$ = this.categoriesSubject.asObservable();
  events$ = this.eventsSubject.asObservable();
  cardsExpandedState$ = this.cardsExpandedStateSubject.asObservable();
  users$ = this.usersSubject.asObservable();
  viewMode$ = this.viewModeSubject.asObservable();
  sortConfig$ = this.sortConfigSubject.asObservable();
  dateFilter$ = this.dateFilterSubject.asObservable();
```

- Plik **validators.ts** – ten plik zawiera w sobie dodatkowe walidatory używane w aplikacji:
    - dateOrder - sprawdzający poprawną kolejność dat
    - dateOverlap - weryfikujący nakładanie się wydarzeń w czasie
    - imageFile - sprawdzający poprawność przesyłanych plików graficznych
    - passwordMatch - porównujący zgodność haseł

```
  static passwordMatch(g: FormGroup): ValidationErrors | null {
    return g.get('password')?.value === g.get('password_confirmation')?.value
      ? null
      : { 'mismatch': true };
  }
}
```

- **app.component** –główny komponent aplikacji
- **app.config.ts** – główny plik konfiguracyjny aplikacji
- **app.routes.ts** – plik routingu, określający ścieżki dostępu do poszczególnych komponentów

```
export const routes: Routes = [
    { path: '', component: HomeComponent },
    { path: 'login', component: LoginComponent },
    { path: 'register', component: RegisterComponent },
    { path: 'change-password', component: ChangePasswordComponent }
  ];
```

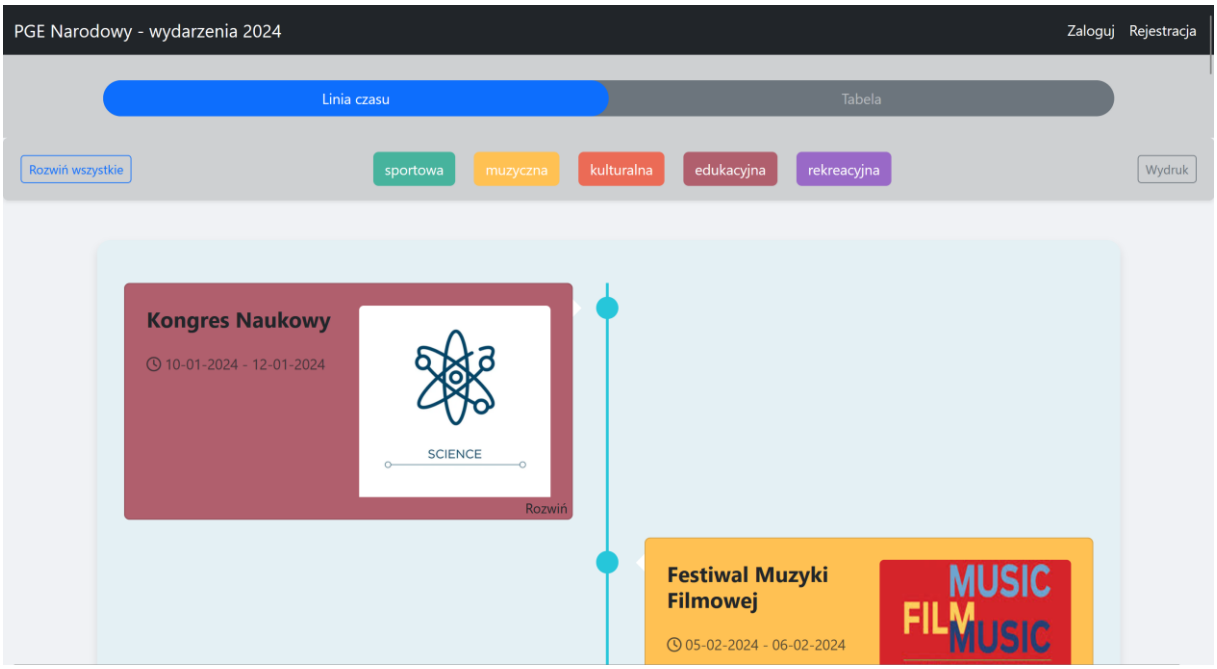
- **package.json** - plik konfiguracyjny projektu Angular, który zawiera metadane projektu oraz listę wszystkich zależności
- **server.js** – plik zawierający prosty serwer Express.js służący do hostowania aplikacji
    -

JS server.js

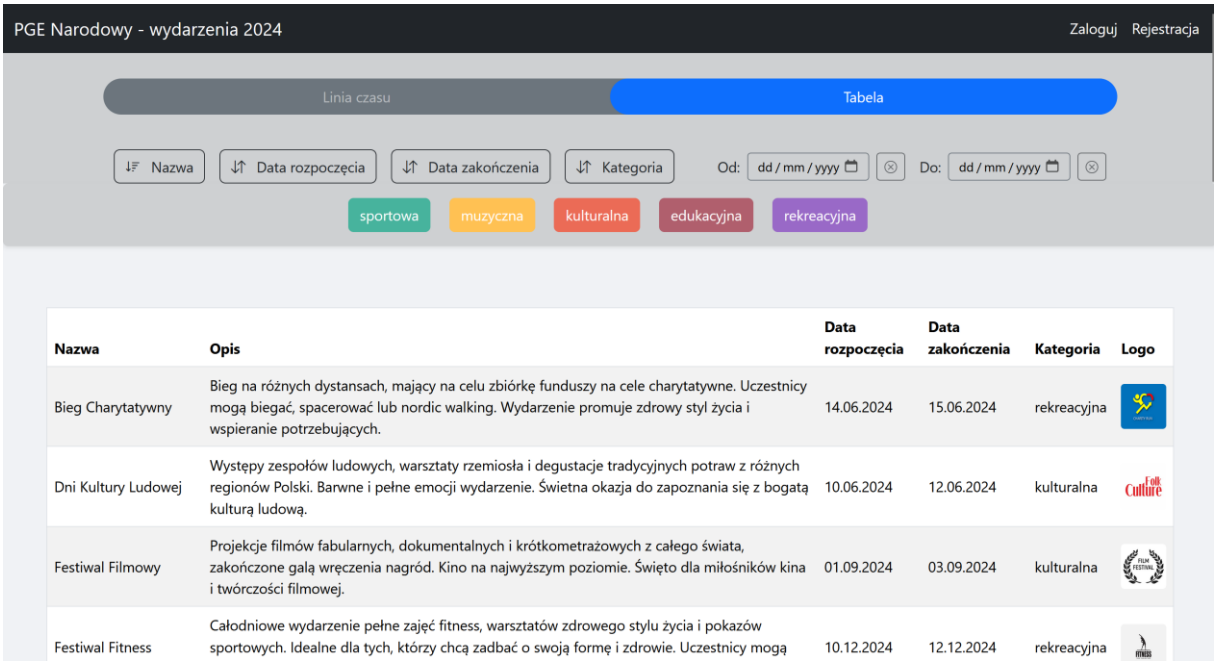
timeline_angular > JS server.js > ...

```
15
16  app.use(express.static(distPath));
17  app.get('/*', function(req, res) {
18      const indexPath = path.join(distPath, 'index.html');
19      console.log('Trying to serve:', indexPath);
20
21      if (require('fs').existsSync(indexPath)) {
22          res.sendFile(indexPath);
23      } else {
24          res.status(404).send('index.html not found. Path: ' + indexPath);
25      }
26  });
27
28  const port = process.env.PORT || 8080;
29  app.listen(port, () => {
30      console.log('Server is running on port', port);
31      console.log('Serving files from:', distPath);
32  });
```

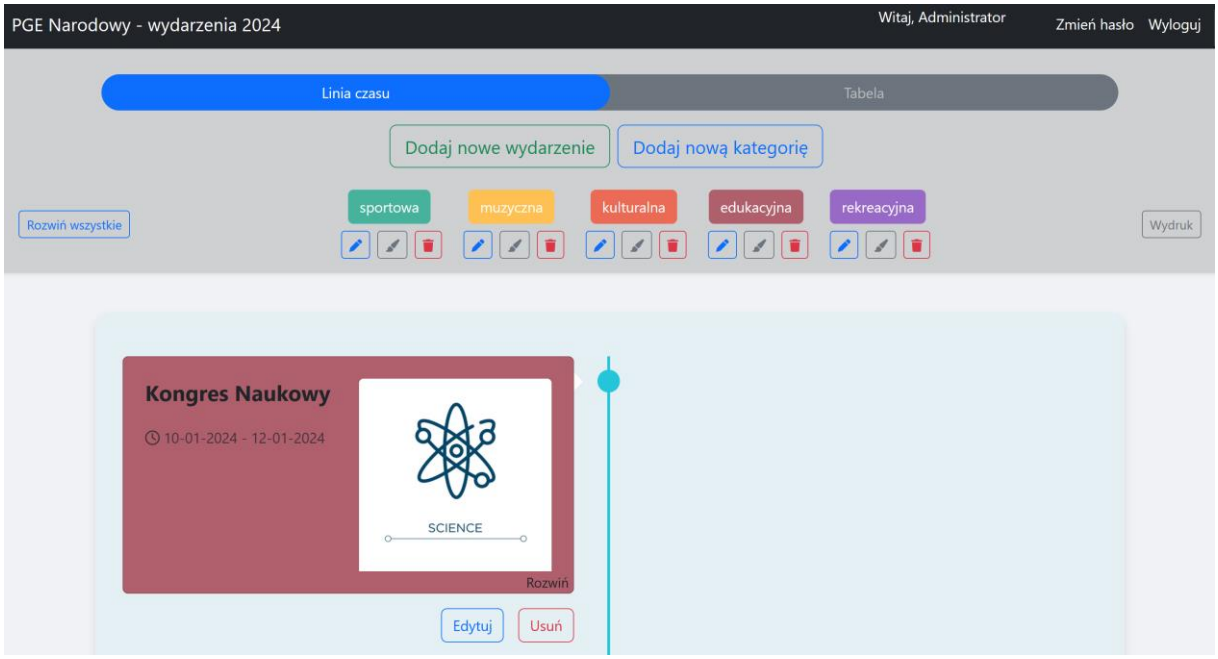
5. Zrzuty ekranu demonstrujące zaimplementowane funkcjonalności



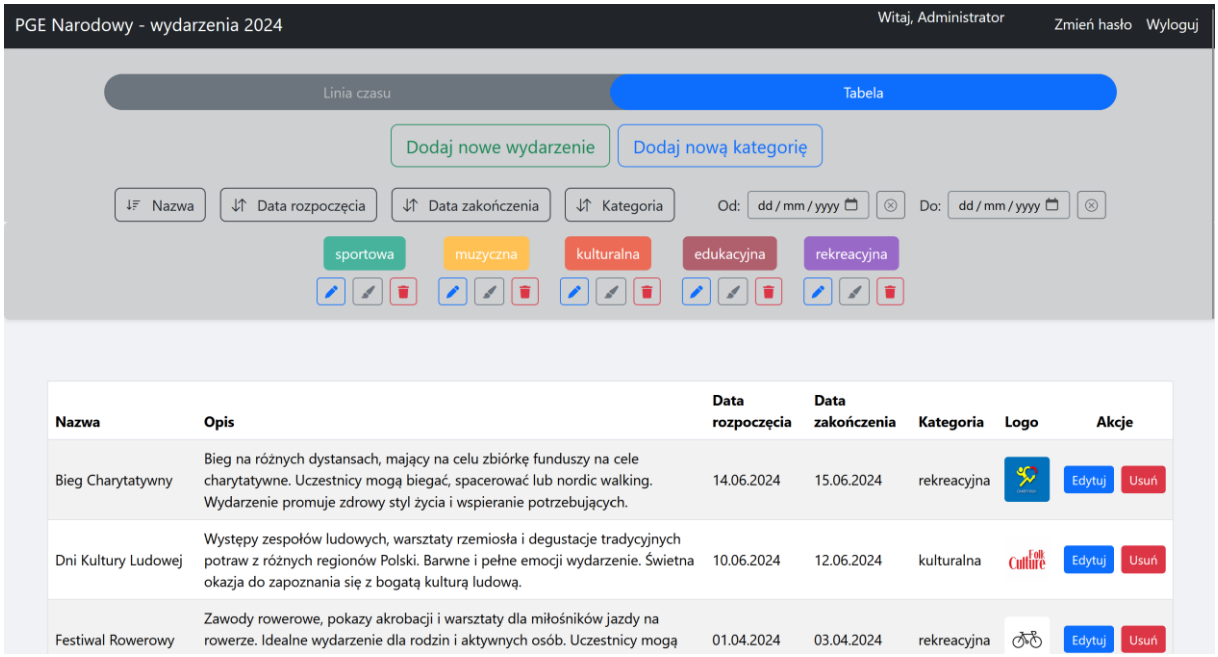
Strona główna – widok linii czasu



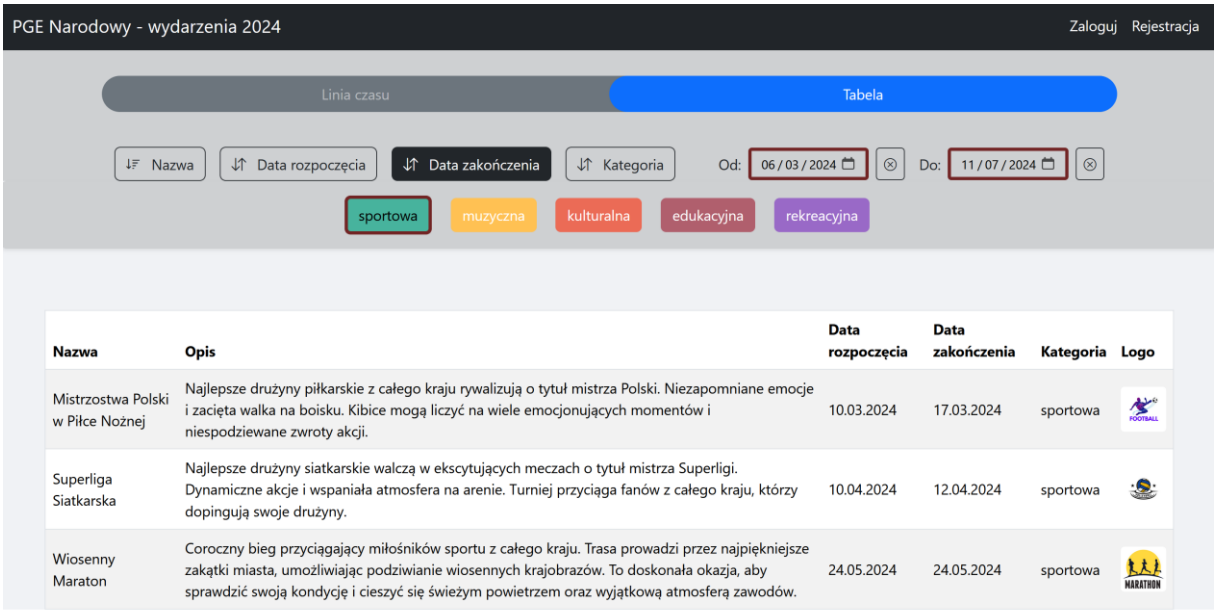
Strona główna – widok tabeli



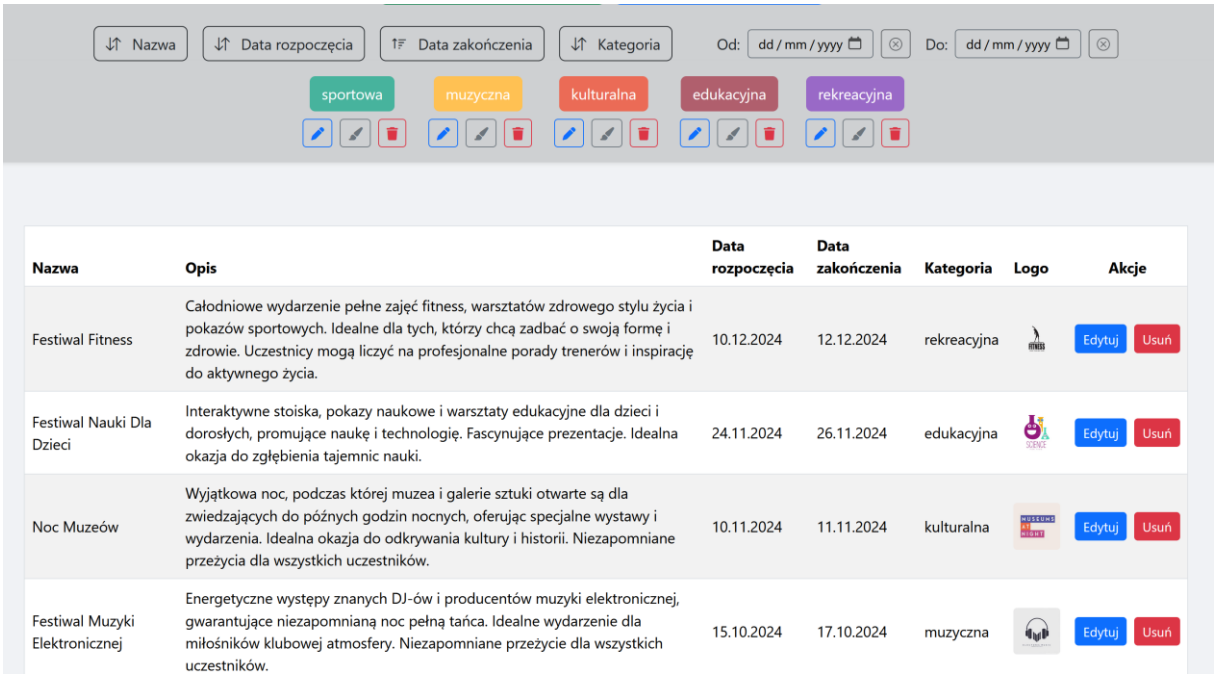
Strona główna (zalogowany użytkownik) – widok osi czasu



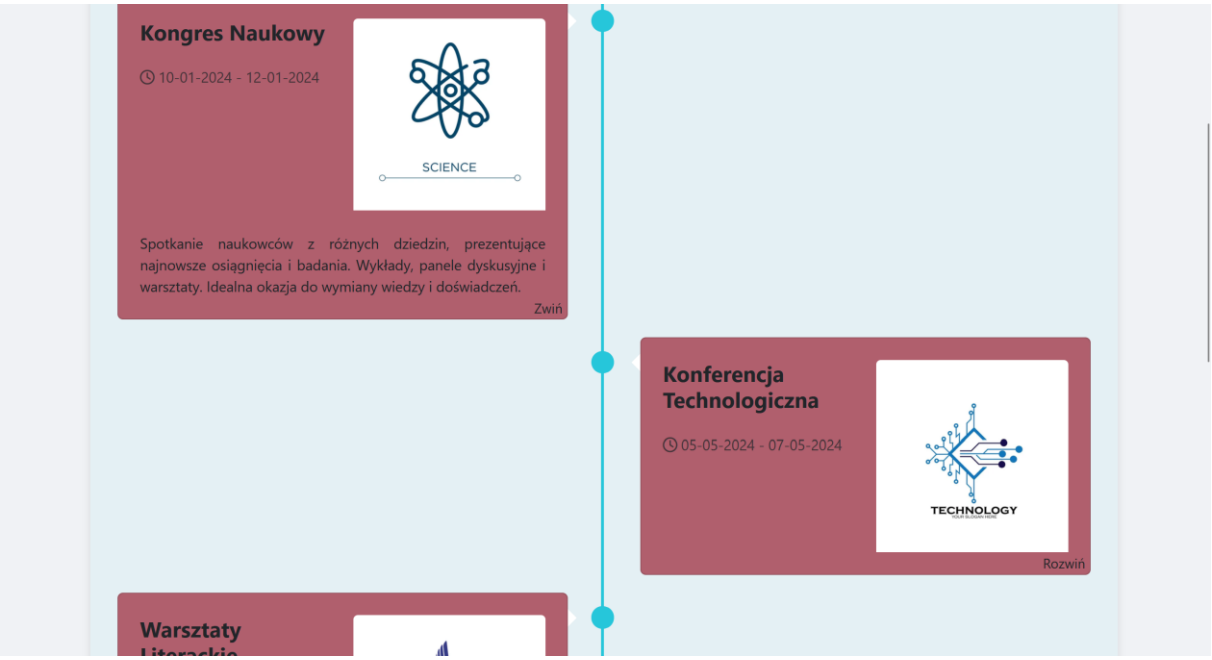
Strona główna (zalogowany użytkownik) – widok tabeli



Wydarzenia przefiltrowane przez kategorie, datę początkowa oraz datę końcową – widok tabeli

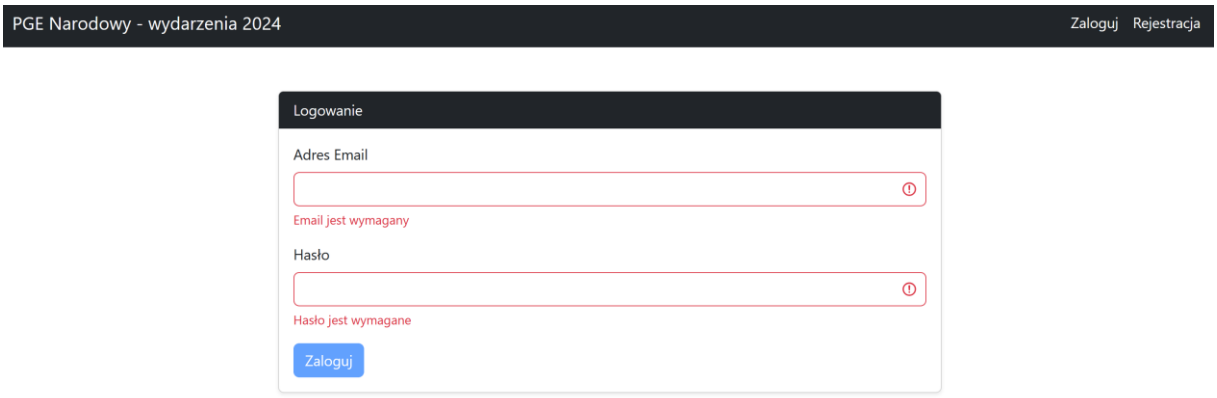


Wydarzenia posortowane po dacie zakończenia (od najnowszej)

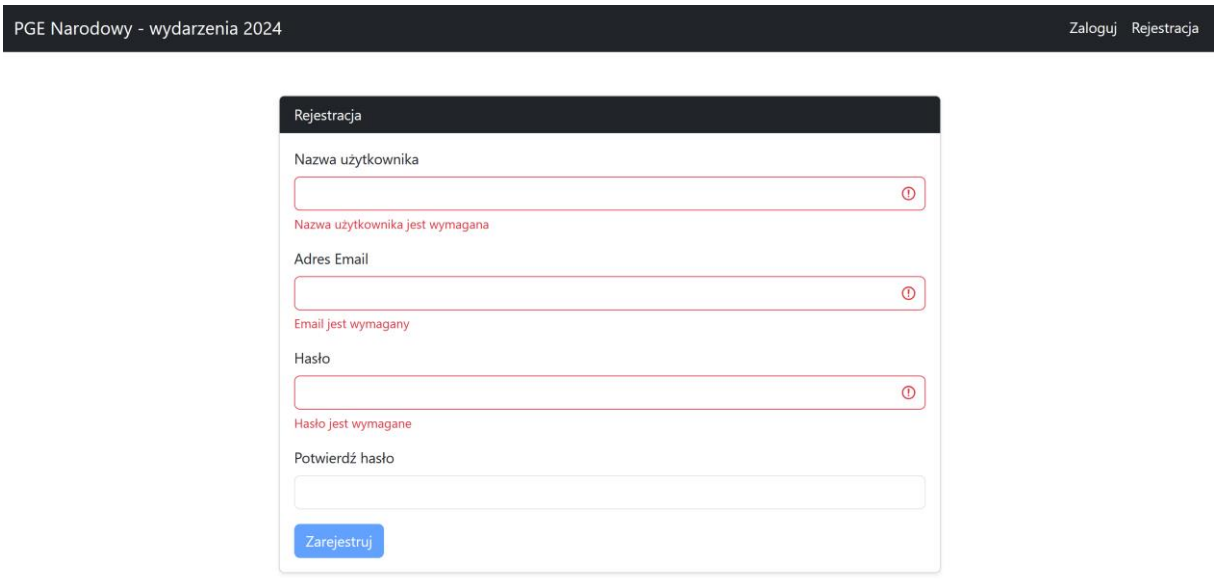


Wydarzenia przefiltrowane przez kategorie– widok osi czasu

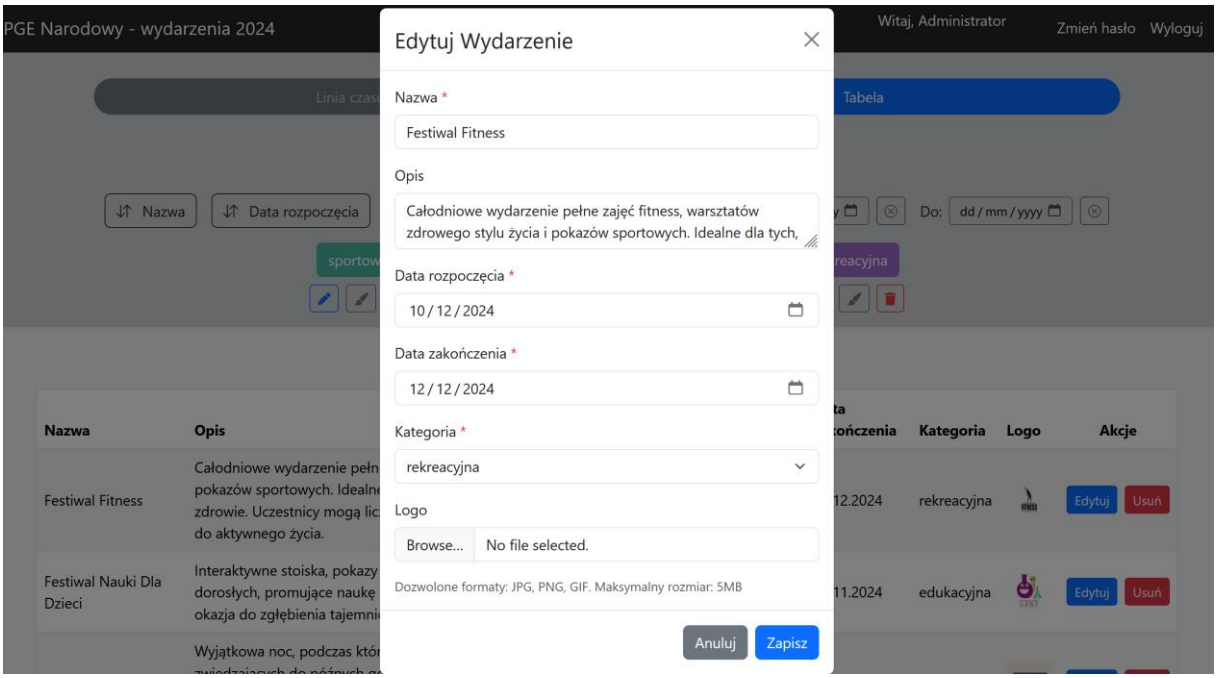
Rozwinięte wydarzenia



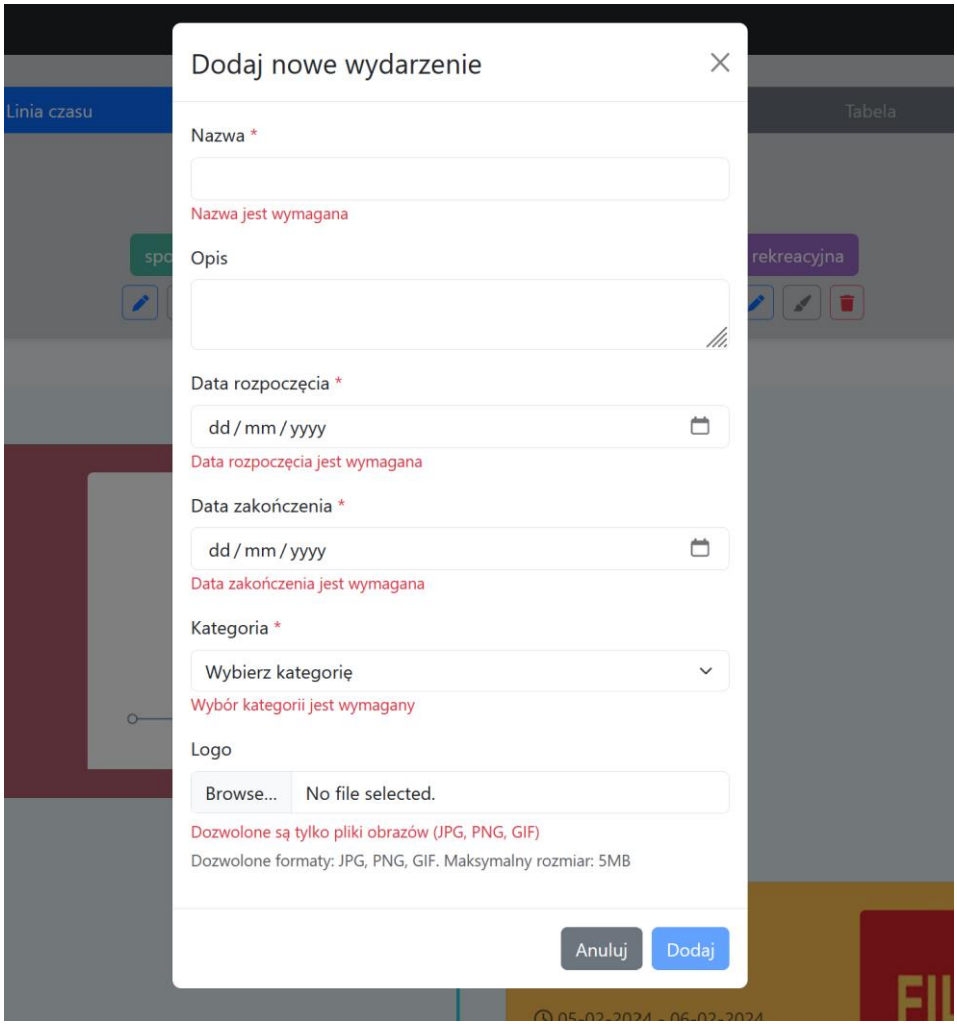
Widok logowania z błędami walidacji



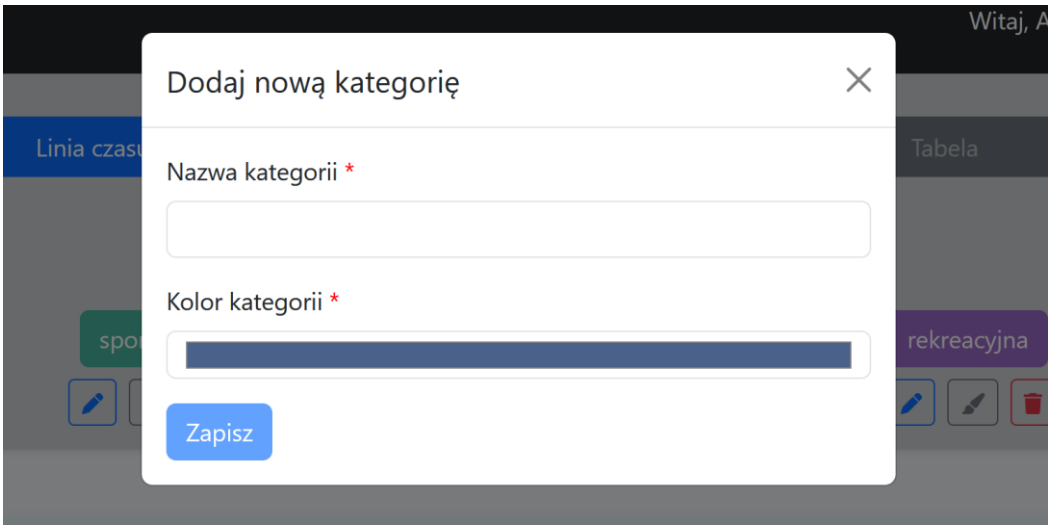
Widok rejestracji z błędami walidacji



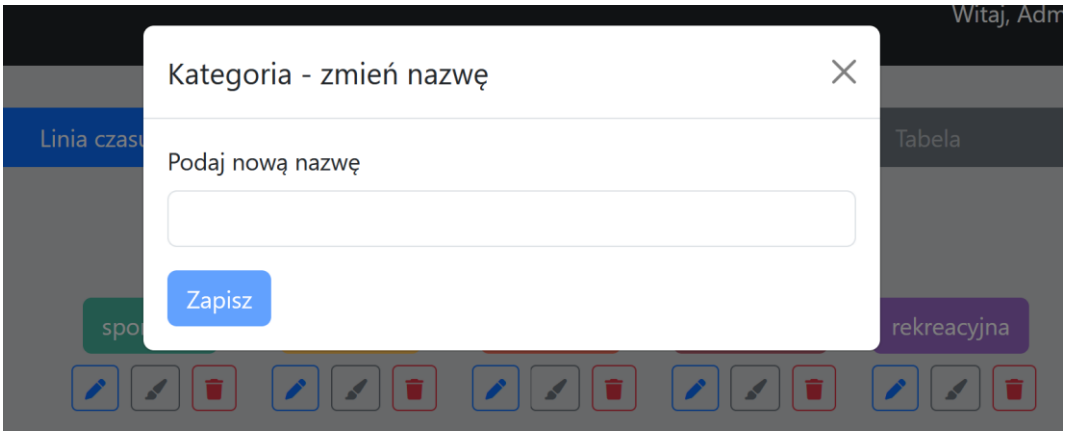
Edycja wydarzenia



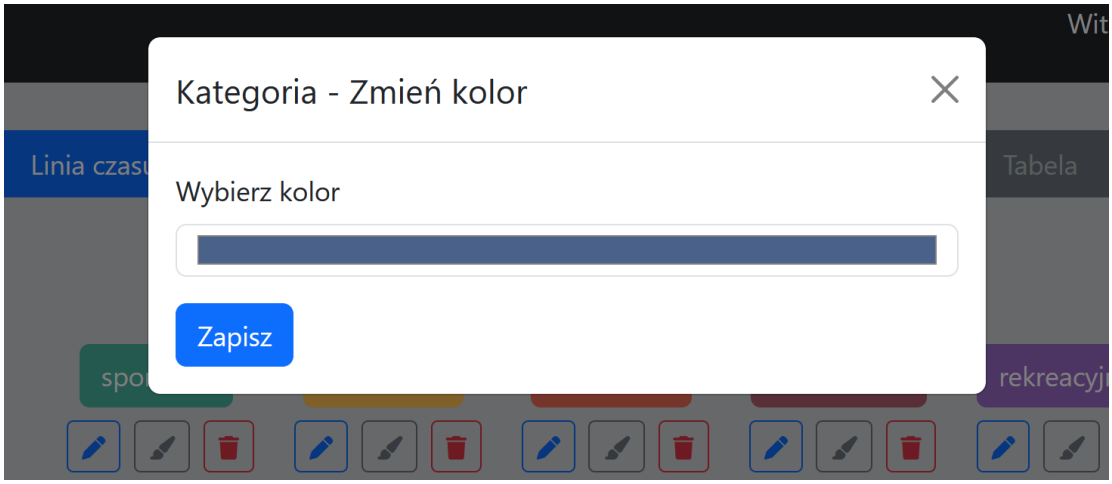
Dodanie nowego wydarzenia z błędami walidacji



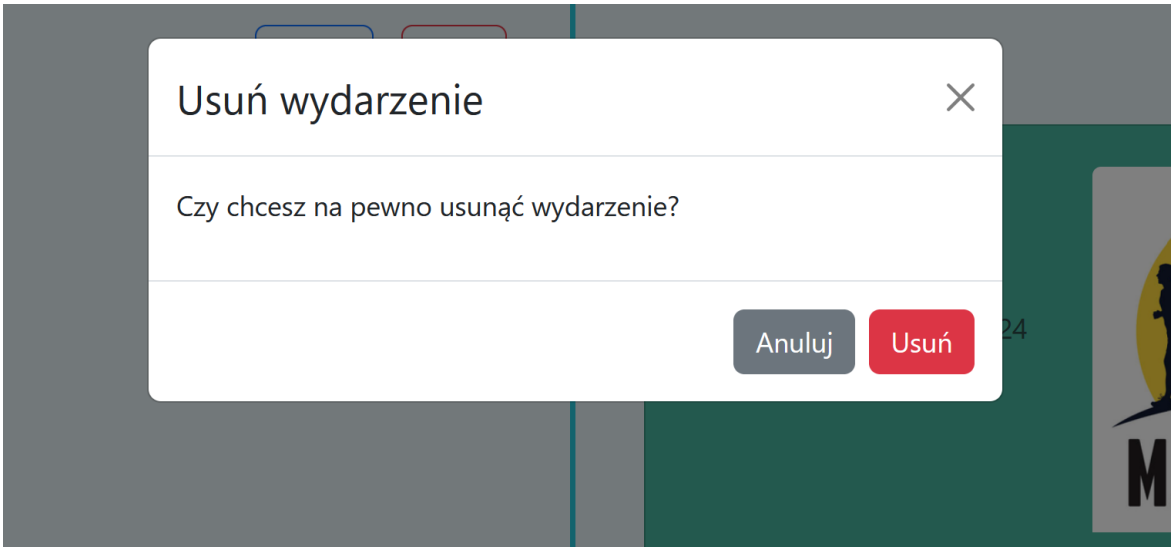
Dodanie nowej kategorii



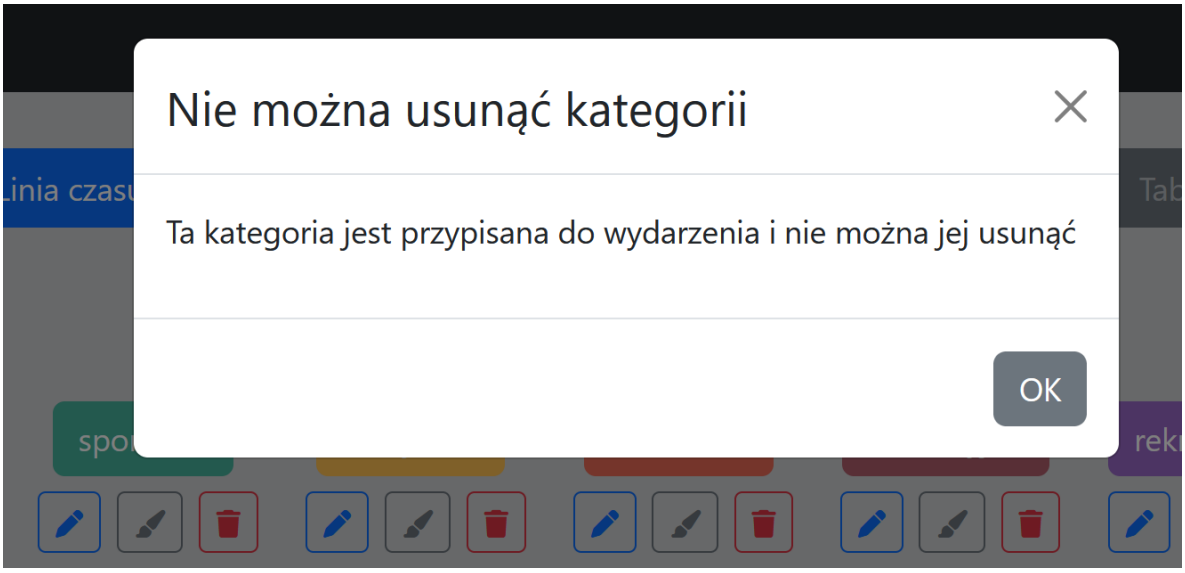
Edycja nazwy kategorii



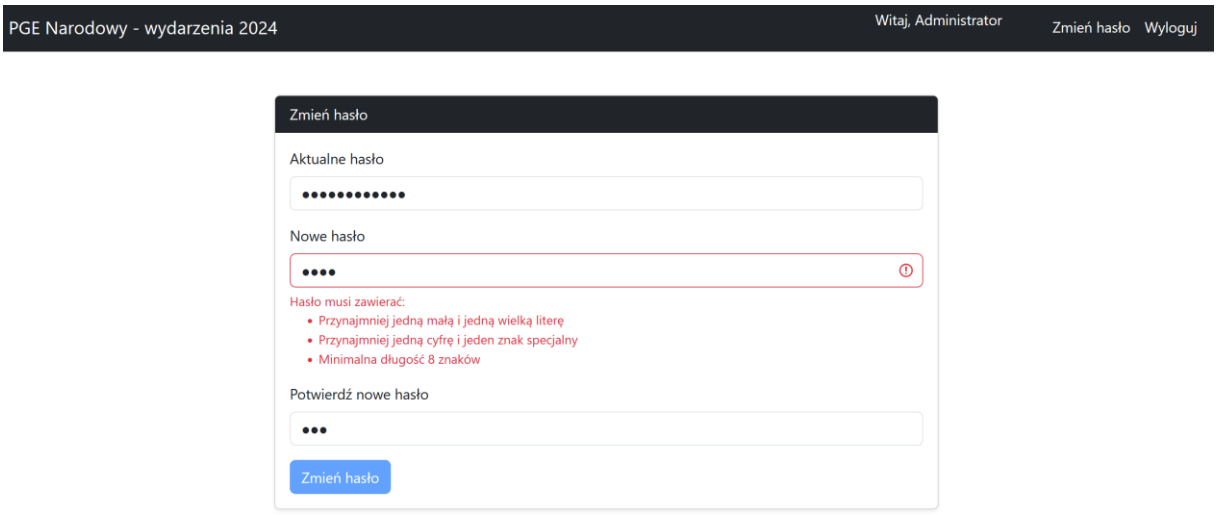
Edycja koloru kategorii



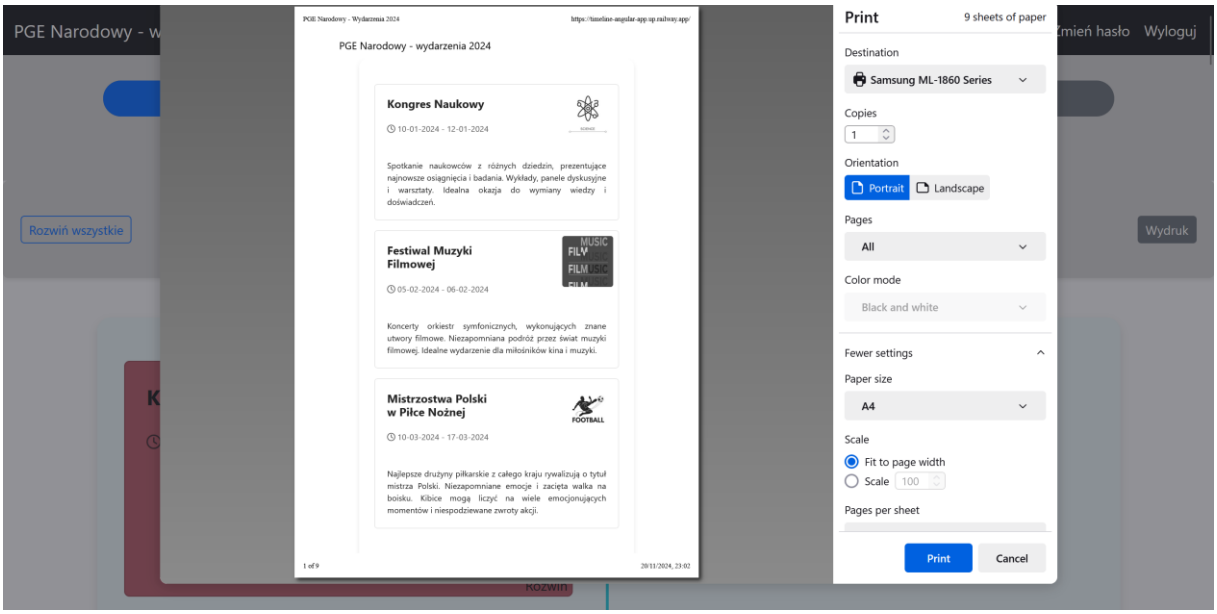
Usuwanie wydarzenia – ekran z potwierdzeniem



Próba usunięcia używanej kategorii – widoczny efekt walidacji



Strona zmiany hasła wraz z błędami walidacji



Strona wydruku