

UAIM

Projekt: Sprawozdanie

Portal rezerwacji wydarzeń "EVE.NT"

Artur Sliepchenko (335900)
Karol Adamski (331155)
Bartłomiej Masiak (331176)
Denys Moldovan (335967)

Politechnika Warszawska, WEITI

3 czerwca 2025

Spis treści

1. Wstęp	3
2. Opis ogólnej koncepcji rozwiązania	3
3. Ogólna architektura systemu	3
4. Lista funkcji: wymagane a zrealizowane	4
5. Aplikacja Webowa	4
5.1. Wprowadzenie i Technologie	4
5.2. Architektura Frontendu	4
5.2.1. Struktura Folderów Projektu	4
5.2.2. Zarządzanie Stanem	5
5.2.3. Routing	5
5.3. Główne Komponenty i Widoki	5
5.3.1. Kluczowe Strony	5
5.3.2. Reużywalne Komponenty UI	6
5.3.3. Komponenty Layoutu	6
5.4. Interakcja z Backendem	7
5.5. Zarządzanie Stanem Użytkownika	7
5.6. System Powiadomień	8
5.7. Efekty działania aplikacji	9
5.7.1. Strona Główna	9
5.7.2. Strona Szczegółów Wydarzenia	11
5.7.3. Formularze Autentykacji	15
5.7.4. Panel Użytkownika	17
5.7.5. Formularz Tworzenia i Edycji Wydarzenia	19
6. Aplikacja Mobilna	22
6.1. Wprowadzenie i Technologie	22
6.2. Architektura Aplikacji	22
6.2.1. Struktura Projektu	22
6.2.2. Nawigacja	23
6.2.3. Zarządzanie Danymi i Stanem	23

6.3.	Główne Komponenty i Widoki	23
6.3.1.	Kluczowe Fragmenty	23
6.3.2.	Elementy List (RecyclerView Adapters)	23
6.4.	Interakcja z Backendem	23
6.5.	Zarządzanie Stanem Użytkownika	23
6.6.	System Filtrowania i Sortowania Wydarzeń	24
6.7.	System Komentarzy	24
6.8.	System Powiadomień	24
6.9.	Struktura Aplikacji i Nawigacja	24
6.10.	Efekty działania aplikacji	26
6.10.1.	Ekran Główny i Lista Wydarzeń	26
6.10.2.	Szczegóły Wydarzenia i Komentarze	27
6.10.3.	Filtrowanie i Sortowanie	28
6.10.4.	Dodawanie i Edycja Wydarzenia	29
6.10.5.	Autentykacja Użytkownika	31
6.10.6.	Profil Użytkownika i Edycja Profilu	32
6.10.7.	System Powiadomień - Działanie	33
6.10.8.	Obsługa Trybu Jasnego i Ciemnego	34
7.	Opis aplikacji back-end	35
7.1.	Architektura Aplikacji	35
7.1.1.	Struktura Projektu	35
7.1.2.	Baza Danych	35
7.1.3.	Autoryzacja i Bezpieczeństwo	35
7.1.4.	Endpointy API	35
7.1.5.	Obsługa Maili	37
7.1.6.	Migracje Bazy Danych	37
7.1.7.	Testowanie	37
7.1.8.	Konteneryzacja	38
8.	Opis Plików Konfiguracyjnych Docker	40
8.1.	Dockerfile dla Aplikacji Backendowej	40
8.2.	Dockerfile dla Aplikacji Frontendowej (Node.js/React)	41
9.	Podsumowanie	42
9.1.	Możliwość Wdrożenia Komercyjnego na Serwerze VPS	42
9.1.1.	Uruchomienie Kontenerów na Serwerze	42
9.1.2.	Udostępnienie w Internecie	42
9.1.3.	Automatyzacja Wdrożenia na Serwerze	43

1. Wstęp

Projekt „**EVE.NT**” – kompleksowy portal do rezerwacji wydarzeń kulturalnych, zaprojektowany i zaimplementowany w ramach przedmiotu *UAIM*. Celem projektu było stworzenie pełnego ekosystemu umożliwiającego użytkownikom przeglądanie, rejestrowanie, organizowanie oraz komentowanie wydarzeń za pośrednictwem zarówno aplikacji webowej, jak i mobilnej.

Projekt opiera się na nowoczesnych technologiach frontendowych i backendowych, a jego realizacja wymagała ścisłej współpracy między członkami zespołu oraz integracji wielu komponentów systemowych. W ramach prac powstały dwie aplikacje: webowa – stworzona przy użyciu React i Vite, oraz mobilna – napisana w języku Java z wykorzystaniem Android SDK.

W kolejnych rozdziałach przedstawiono szczegóły implementacji, architekturę systemu, zastosowane technologie oraz funkcjonalności udostępniane użytkownikowi, a także efekty końcowe w postaci działających interfejsów.

2. Opis ogólnej koncepcji rozwiązania

Projekt **EVE.NT** stanowi kompleksowy system do obsługi wydarzeń kulturalnych, umożliwiający użytkownikom rejestrację, przeglądanie, tworzenie, rezerwację oraz komentowanie wydarzeń. System składa się z trzech głównych komponentów:

- aplikacji webowej stworzonej w technologii React + Vite,
- aplikacji mobilnej natywnej na system Android zbudowanej w języku Java,
- aplikacji backendowej REST API zrealizowanej w Pythonie z użyciem Flask.

Głównym celem było zapewnienie wieloplatformowej dostępności i spójnego doświadczenia użytkownika oraz pełnej funkcjonalności niezależnie od rodzaju urządzenia. Dodatkowo projekt uwzględnia aspekty bezpieczeństwa, modularności oraz możliwości łatwego wdrożenia produkcyjnego z użyciem Dockera.

3. Ogólna architektura systemu

System zbudowany jest w architekturze klient-serwer, z jasno wyodrębnioną warstwą prezentacji (frontend), logiki biznesowej (backend) i warstwy danych (baza PostgreSQL). Schemat architektury:

- **Frontend Web:** aplikacja w React z modularną strukturą folderów, zarządzaniem stanem przy użyciu React Context API i komunikacją z backendem poprzez API.
- **Frontend Mobile:** aplikacja Android oparta o Fragmenty i Navigation Component, komunikacja z backendem przez HttpURLConnection oraz AsyncTask.
- **Backend:** REST API w Flask, modularne blueprinty, autoryzacja oparta o JWT, ORM SQLAlchemy, obsługa maili (Flask-Mail), konfiguracja z pliku .env.
- **Baza danych:** PostgreSQL z modelami użytkowników, wydarzeń, rezerwacji, komentarzy, lokalizacji, kategorii oraz relacjami wiele-do-wielu.
- **Konteneryzacja:** Docker + docker-compose umożliwiające łatwe wdrożenie backendu, frontendu oraz bazy danych na serwerze VPS.

4. Lista funkcji: wymagane a zrealizowane

Funkcja	Status
Rejestracja i logowanie	Zrealizowano
Tworzenie i edycja wydarzeń	Zrealizowano
Rezerwacja miejsc	Zrealizowano
System komentarzy	Zrealizowano
Panel użytkownika	Zrealizowano
Filtры i sortowanie wydarzeń	Zrealizowano częściowo
Powiadomienia	Zrealizowano
Tryb ciemny/jasny	Zrealizowano
Wysyłka przypomnień mailowych	Zrealizowano

5. Aplikacja Webowa

5.1. Wprowadzenie i Technologie

Głównym celem aplikacji webowej jest umożliwienie użytkownikom przeglądania, rezerwowania oraz zarządzania wydarzeniami kulturalnymi. Interfejs został zaprojektowany z myślą o intuicyjnej nawigacji i łatwym dostępie do kluczowych funkcjonalności, takich jak rejestracja, logowanie, przeglądanie szczegółów wydarzeń oraz zarządzanie własnym profilem i rezerwacjami.

Do budowy aplikacji webowej wykorzystano następujące technologie i narzędzia:

- **React:** Biblioteka JavaScript do budowy interfejsów użytkownika, umożliwiająca tworzenie reużywalnych komponentów i efektywne zarządzanie stanem.
- **Vite:** Nowoczesne narzędzie do budowy frontendu, zapewniające szybki start serwera deweloperskiego oraz zoptymalizowaną budowę produkcyjną.
- **React Router DOM:** Biblioteka do obsługi routingu po stronie klienta, umożliwiająca nawigację między różnymi widokami aplikacji bez przeładowywania całej strony.
- **JavaScript:** Główny język programowania aplikacji.
- **CSS Modules:** Podejście do stylowania komponentów, zapewniające lokalny zasięg klas CSS i unikające konfliktów nazw. Style są definiowane w plikach `.module.css` powiązanych z konkretnymi komponentami.
- **Zmienne CSS:** Używane do zarządzania globalną paletą kolorów i innymi wartościami stylistycznymi.
- **Google Material Symbols:** Zestaw ikon wykorzystywany do poprawy wizualnej i intuicyjności interfejsu.

5.2. Architektura Frontendu

Aplikacja webowa została zaimplementowana zgodnie z modelem Single Page Application, gdzie większość zasobów jest ładowana jednorazowo, a nawigacja i aktualizacje interfejsu odbywają się dynamicznie po stronie klienta.

5.2.1. Struktura Folderów Projektu

Projekt frontendowy posiada zorganizowaną strukturę folderów w katalogu `src/`, mającą na celu ułatwienie nawigacji, utrzymania i skalowalności kodu:

- `components/`: Zawiera reużywalne komponenty UI oraz komponenty specyficzne dla poszczególnych modułów.
 - `UI/`: Generyczne, prezentacyjne komponenty interfejsu, takie jak `Button`, `Input` i `Select`.
 - `Layout/`: Komponenty odpowiedzialne za główny układ strony: `MainLayout`, `Navbar` i `Footer`.
- Pozostałe podfoldery grupują komponenty związane z konkretnymi funkcjonalnościami, np. `Events/` i `Location/`.

- **contexts/**: Przechowuje implementacje React Context API, takie jak **AuthContext** do zarządzania stanem autentykacji oraz **AlertContext** do globalnych powiadomień.
- **pages/**: Zawiera komponenty reprezentujące poszczególne strony aplikacji, które są mapowane na konkretne ścieżki URL przez router. Przykładowe strony to **HomePage**, **LoginPage**, **EventDetailPage** i **UserPage**.
- **services/**: Moduły odpowiedzialne za komunikację z API backendu. Każdy serwis grupuje funkcje wykonujące żądania HTTP do odpowiednich endpointów.
- **palette.css**: Globalna paleta kolorów.
- **App.jsx**: Główny komponent aplikacji, w którym konfigurowany jest routing i globalne provider'y kontekstu.
- **main.jsx**: Punkt wejścia aplikacji React, inicjalizujący renderowanie korzenia aplikacji oraz **BrowserRouter**.

5.2.2. Zarządzanie Stanem

Do zarządzania stanem globalnym aplikacji, takim jak informacje o zalogowanym użytkowniku oraz system powiadomień, wykorzystano mechanizm React Context API.

- **AuthContext**: Odpowiada za przechowywanie danych zalogowanego użytkownika i statusu autentykacji. Udostępnia funkcje `tryLogin()` i `logout()`.
- **AlertContext**: Umożliwia wyświetlanie globalnych powiadomień o sukcesie, błędzie czy informacji, poprzez funkcję `showAlert()`. Komponent **AlertDisplay** konsumuje ten kontekst do renderowania alertów.

Stan lokalny komponentów zarządzany jest przy użyciu hooków `useState` i `useEffect`.

5.2.3. Routing

Nawigacja w aplikacji realizowana jest za pomocą biblioteki **React Router DOM**. Główne trasy zdefiniowane są w komponencie **App.jsx**. Wykorzystywany jest komponent `<Routes>` do definiowania poszczególnych ścieżek i mapowania ich na odpowiednie komponenty stron z katalogu **pages/**. Przykładowe trasy to:

- `/`: Strona główna.
- `/login`, `/register`: Strony autentykacji.
- `/events/:eventId`: Szczegóły konkretnego wydarzenia.
- `/events/create`: Formularz tworzenia nowego wydarzenia.
- `/events/edit/:eventId`: Formularz edycji istniejącego wydarzenia.
- `/profile`: Strona profilu zalogowanego użytkownika.
- `/profile/edit`: Formularz edycji profilu użytkownika.

5.3. Główne Komponenty i Widoki

Aplikacja webowa składa się z szeregu komponentów-stron, które realizują poszczególne funkcjonalności dostępne dla użytkownika, oraz z reużywalnych komponentów UI, budujących interfejs.

5.3.1. Kluczowe Strony

Komponenty stron są umieszczone w katalogu `src/pages/` i odpowiadają za logikę oraz prezentację poszczególnych widoków aplikacji.

- **Strona Główna**: Pierwszy widok po wejściu do aplikacji. Prezentuje ogólne informacje o portalu oraz listę wszystkich dostępnych wydarzeń publicznych, pobieranych za pomocą serwisu `events.js`. Wykorzystuje komponent `EventList` do wyświetlania wydarzeń.
- **Strona Logowania**: Umożliwia użytkownikom zalogowanie się do systemu przy użyciu nazwy użytkownika lub adresu email oraz hasła. Po pomyślnym zalogowaniu, użytkownik jest przekierowywany, a jego dane są zapisywane w `localStorage` i aktualizowany jest **AuthContext**.
- **Strona Rejestracji**: Pozwala nowym użytkownikom na założenie konta. Formularz zawiera nazwę użytkownika, imię, nazwisko, datę urodzenia, email, numer telefonu i hasło.

- **Strona Szczegółów Wydarzenia:** Wyświetla pełne informacje o wybranym wydarzeniu, pobierane na podstawie ID wydarzenia z URL. Prezentuje tytuł, opis, datę, imiona organizatora, lokalizację, liczbę uczestników, a także listę komentarzy. Umożliwia zalogowanym użytkownikom rezerwację miejsca (jeśli nie są organizatorami i są wolne miejsca) lub anulowanie rezerwacji. Pozwala także na dodawanie, edycję i usuwanie własnych komentarzy. Organizatorzy wydarzenia mają możliwość jego usunięcia lub przejścia do edycji.
- **Strona Tworzenia Wydarzenia:** Dostępna dla zalogowanych użytkowników. Strona ta renderuje reużywalny komponent formularza `EventForm` w trybie tworzenia. Umożliwia zdefiniowanie wszystkich niezbędnych danych nowego wydarzenia, w tym wybór lub dynamiczne dodanie lokalizacji i kategorii.
- **Strona Edycji Wydarzenia:** Dostępna dla organizatora danego wydarzenia. Pobiera ID wydarzenia z URL, wczytuje jego aktualne dane i renderuje komponent `EventForm` w trybie edycji, wypełniając go istniejącymi informacjami.
- **Strona Profilu Użytkownika:** Dostępna dla zalogowanego użytkownika. Wyświetla jego dane osobowe oraz listę wydarzeń, których jest organizatorem lub uczestnikiem. Zawiera przyciski umożliwiające edycję profilu oraz dodanie nowego wydarzenia. Dodatkowo, prezentuje sekcje do zarządzania własnymi lokalizacjami i kategoriami.
- **Strona Edycji Profilu:** Umożliwia zalogowanemu użytkownikowi modyfikację swoich danych osobowych.
- **Strona 404:** Wyświetlana, gdy użytkownik spróbuje uzyskać dostęp do nieistniejącej ścieżki URL.

5.3.2. Reużywalne Komponenty UI

W celu zapewnienia spójności interfejsu i unikania duplikacji kodu, stworzono szereg generycznych komponentów UI, znajdujących się w katalogu `src/components/UI/`:

- **Button.jsx:** Reużywalny komponent przycisku z możliwością konfiguracji wariantów i stanu.
- **Input.jsx:** Komponent pola tekstowego z obsługą etykiety, placeholdera, typu inputu i wyświetlania błędów walidacyjnych.
- **Textarea.jsx:** Komponent wieloliniowego pola tekstopowego, podobny w funkcjonalności do `Input.jsx`.
- **Select.jsx:** Komponent listy rozwijanej do wyboru opcji, z obsługą etykiety i błędów.
- **Modal.jsx:** Generyczny komponent okna modalnego, renderowany za pomocą React Portals, używany do wyświetlania formularzy nad resztą treści strony.
- **EventCard.jsx:** Komponent karty prezentującej skrócone informacje o pojedynczym wydarzeniu, używany na listach wydarzeń. Zawiera logikę formatowania danych i warunkowego wyświetlania informacji, np. dla właściciela wydarzenia.
- **EventList.jsx:** Komponent odpowiedzialny za pobieranie i wyświetlanie listy wydarzeń w formie siatki komponentów `EventCard`. Obsługuje stany ładowania i błędów.
- **DeleteGrid.jsx:** Reużywalny komponent tabelaryczny wyświetlający listę elementów z opcją ich usunięcia.
- **AlertDisplay.jsx:** Komponent wyświetlający globalne powiadomienia na podstawie danych z `AlertContext`.

5.3.3. Komponenty Layoutu

Struktura wizualna większości stron aplikacji jest zarządzana przez komponenty layoutu, znajdujące się w katalogu `src/components/Layout/`:

- **MainLayout.jsx:** Główny kontener aplikacji, który opakowuje zawartość poszczególnych stron. Renderuje komponenty `Navbar` i `Footer` oraz dynamicznie zarządza górnym marginesem głównej treści, aby uwzględnić pozycjonowanie `Navbar`.
- **Navbar.jsx:** Pasek nawigacyjny aplikacji, wyświetlający logo portalu oraz linki do głównych sekcji. Jego wygląd dynamicznie dostosowuje się w zależności od aktualnej strony oraz pozycji scrolla. Wyświetla również linki związane z autentykacją na podstawie stanu użytkownika z `AuthContext`.
- **Footer.jsx:** Stopka aplikacji, zawierająca informacje o prawach autorskich.

5.4. Interakcja z Backendem

Komunikacja frontendowej części aplikacji z backendem odbywa się za pomocą żądań HTTP kierowanych do zdefiniowanych endpointów API. Cała logika tej komunikacji została hermetyzowana w dedykowanych modułach znajdujących się w katalogu `src/services/`.

Główne moduły serwisowe to:

- **auth.js:** Zawiera funkcje wspomagające obsługę zapytań HTTP — w szczególności `handleResponse` do jednolitej obsługi odpowiedzi i błędów z serwera, `getAuthHeaders` do tworzenia nagłówków z tokenem dostępu, oraz `getUserHeaders` do dodawania UUID użytkownika do nagłówków.
- **register.js:** Odpowiedzialny za validację danych rejestracyjnych po stronie klienta oraz za rejestrację użytkownika poprzez funkcję `tryRegister`.
- **login.js:** Obsługuje logowanie użytkownika.
- **user.js:** Oferuje funkcje do pobierania i aktualizacji danych zalogowanego użytkownika.
- **events.js:** Zawiera funkcje do obsługi wydarzeń, takie jak pobieranie listy wydarzeń, filtrowanie po kategoriach, pobieranie wydarzeń użytkownika, tworzenie, edycja i usuwanie wydarzeń oraz pobieranie szczegółów pojedynczego wydarzenia.
- **category.js:** Odpowiedzialny za operacje na kategoriach wydarzeń.
- **eventCategory.js:** Zarządza relacjami wiele-do-wielu między wydarzeniami a kategorią, udostępniając funkcje takie jak `linkEventToCategory`, `unlinkEventFromCategory` oraz `getAllEventCategoryRelations`.
- **location.js:** Obsługuje lokalizacje wydarzeń — umożliwia pobieranie, dodawanie, przeglądanie szczegółów oraz usuwanie lokalizacji.
- **reservation.js:** Serwis do tworzenia i zarządzania rezerwacjami użytkowników.
- **comment.js:** Odpowiada za obsługę komentarzy użytkowników do wydarzeń — ich dodawanie, edycję, pobieranie i usuwanie.
- **translate.js:** Zawiera funkcje przekształcające odpowiedzi zwracane przez backend na przyjazne użytkownikowi komunikaty w języku polskim. Nie jest oparty na faktycznym tłumaczeniu, ale na kodach błędu i strukturze odpowiedzi backendu.

Wszystkie żądania do API wykorzystują natywny interfejs `fetch`. W środowisku deweloperskim, aby uniknąć problemów związanych z CORS, skonfigurowano serwer proxy w pliku `vite.config.js`. Przekierowuje on żądania z frontendu (np. z `/api/...`) na odpowiedni port serwera backendowego.

5.5. Zarządzanie Stanem Użytkownika

Mechanizm autentykacji i zarządzania sesją użytkownika jest kluczowym elementem aplikacji, zrealizowanym przy użyciu React Context API oraz interakcji z odpowiednimi endpointami backendu.

Centralnym punktem zarządzania stanem użytkownika jest **AuthContext**. **AuthProvider**, opakowujący główną aplikację, jest odpowiedzialny za:

- Przechowywanie informacji o aktualnie zalogowanym użytkowniku, w tym jego danych oraz statusu autentyczacji.
- Zarządzanie stanem ładowania informacji o użytkowniku, co pozwala na wyświetlanie odpowiednich komunikatów lub placeholderów w interfejsie podczas weryfikacji sesji.
- Udostępnianie funkcji `tryLogin`: wywoływanej po pomyślnym zalogowaniu przez komponent `LoginPage.jsx`. Funkcja ta zapisuje otrzymane tokeny (Access Token i Refresh Token) oraz UUID użytkownika w `localStorage` przeglądarki i aktualizuje stan `currentUser` w kontekście.
- Udostępnianie funkcji `logout()`: odpowiedzialnej za wylogowanie użytkownika. Usuwa ona tokeny i UUID użytkownika z `localStorage` oraz resetuje stan `currentUser` w kontekście na `null`.
- Automatyczne sprawdzanie statusu użytkownika przy inicjalizacji aplikacji: jeśli w `localStorage` znajduje się Access Token i User UUID, wywoływana jest funkcja serwisowa `getUser()`, która próbuje pobrać dane użytkownika z backendu na podstawie tokenu. W przypadku sukcesu, stan `currentUser` jest aktualizowany. W przypadku błędu (np. token wygasł), tokeny są usuwane, a użytkownik traktowany jest jako niezalogowany.

- Udostępnianie funkcji `refreshAuthStatus()`, która może być użyta do ponownego sprawdzenia i odświeżenia danych zalogowanego użytkownika.

Dostęp do chronionych zasobów i stron (np. profil użytkownika, tworzenie/edykcja wydarzeń) jest warunkowany stanem `isAuthenticated` z `AuthContext`. Komponenty te sprawdzają ten stan i w przypadku braku autentykacji, użytkownik jest przekierowywany na stronę logowania za pomocą hooka `useNavigate` z React Router DOM.

Tokeny JWT (Access Token) są wysyłane w nagłówku `Authorization` jako `bearer <token>` przy każdym chronionym żądaniu do API, co jest realizowane przez funkcję pomocniczą `getAuthHeaders()` w `auth.js`.

5.6. System Powiadomień

W celu informowania użytkownika o wynikach operacji jak pomyślne utworzenie wydarzenia, zaimplementowano globalny system powiadomień oparty na React Context API.

System składa się z:

- **AlertContext:** Zarządza tablicą aktywnych alertów. Każdy alert jest obiektem zawierającym unikalne ID generowane przez prosty licznik, treść wiadomości, typ alertu oraz czas trwania.
- **AlertProvider:** Komponent dostawcy kontekstu, który opakowuje aplikację i udostępnia:
 - Funkcję `showAlert(message, type, duration)`: Pozwala dowolnemu komponentowi na wywołanie nowego alertu. Po określonym czasie (`defaultTimeout` lub przekazany `duration`), alert jest automatycznie usuwany.
 - Funkcję `removeAlert(id)`: Umożliwia ręczne usunięcie alertu.
 - Stan `alerts`: Tablica aktualnie wyświetlanych alertów.
- **Hook `useAlert()`:** Uproszczony dostęp do funkcji `showAlert` i `removeAlert` z kontekstu.
- **AlertDisplay.jsx:** Komponent UI, który konsumuje `AlertContext` za pomocą hooka `useAlert()`. Jest on renderowany w głównym komponencie aplikacji i odpowiada za wyświetlanie alertów na ekranie, w prawym górnym rogu widoku. Każdy alert posiada przycisk zamknięcia. Wykorzystuje React Portals do renderowania poza głównym drzewem DOM komponentu `App`.

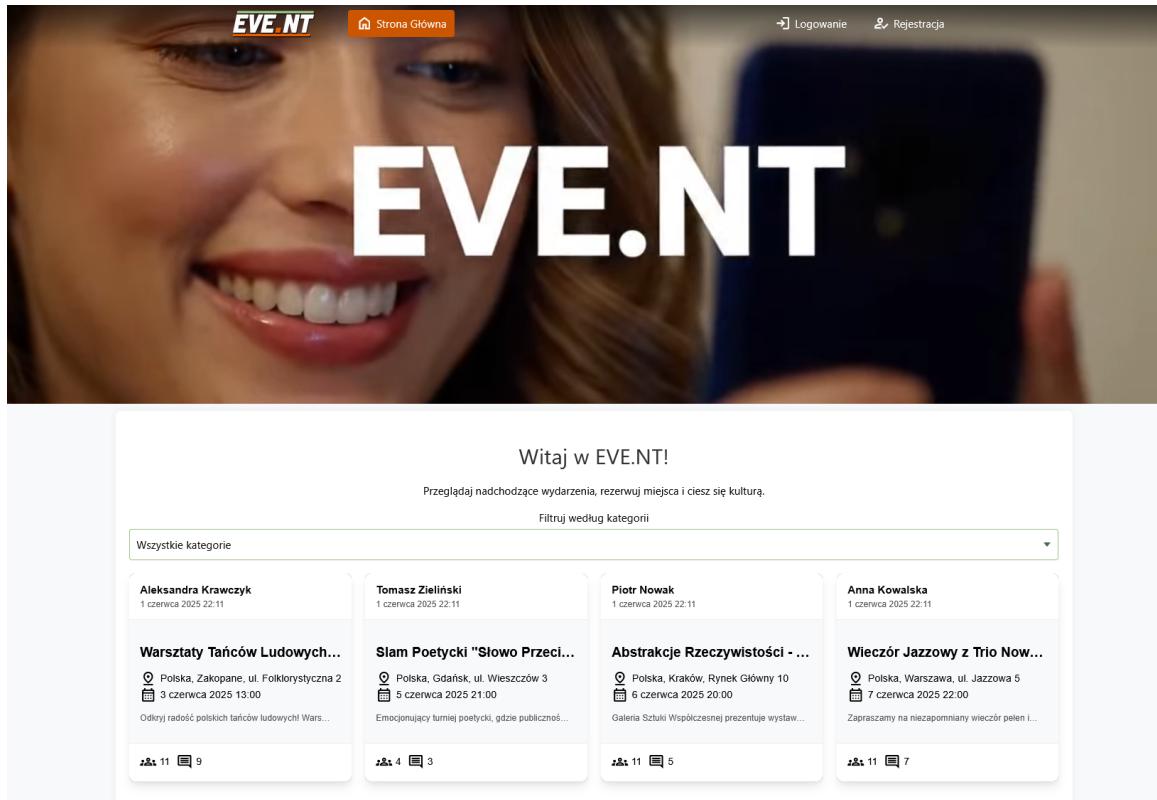
Dzięki temu podejściu, komponenty takie jak formularze logowania, rejestracji czy tworzenia wydarzeń mogą łatwo informować użytkownika o rezultacie swoich działań w spójny i nieinwazyjny sposób, zastępując standarde okienka `alert()` przeglądarki.

5.7. Efekty działania aplikacji

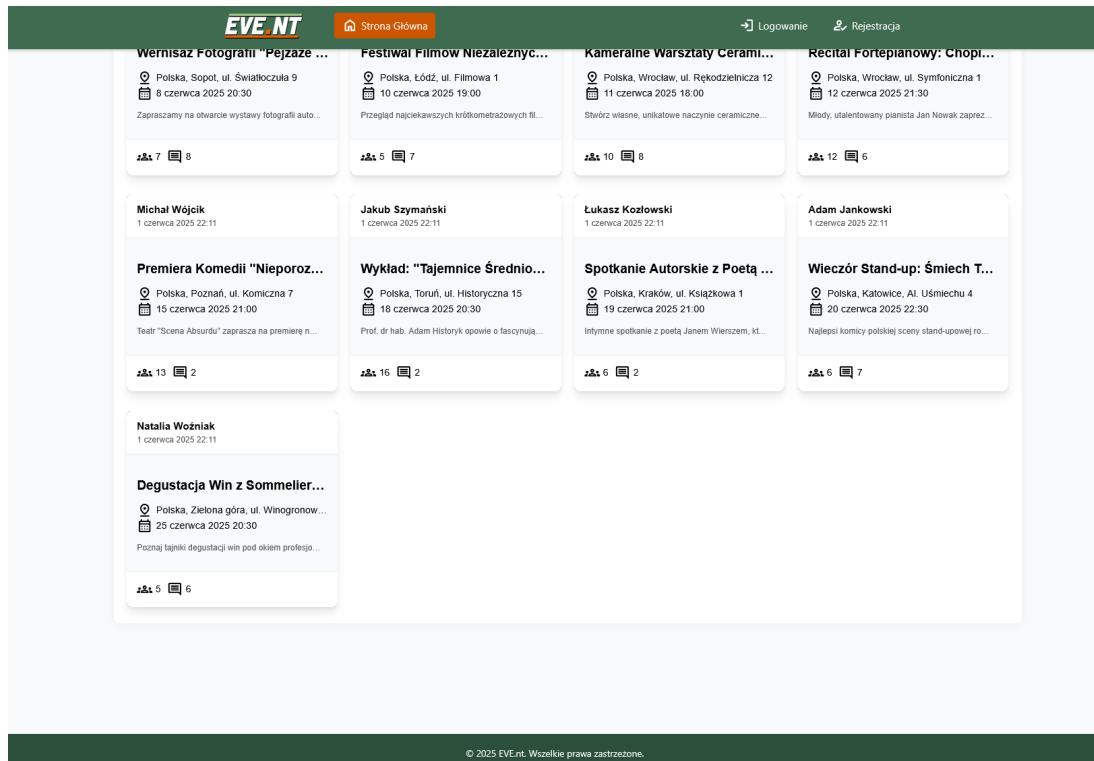
W tej sekcji zaprezentowano wygląd kluczowych elementów interfejsu użytkownika aplikacji webowej EVE.NT. Poniższe zrzuty ekranu ilustrują główne widoki i funkcjonalności dostępne dla użytkowników.

5.7.1. Strona Główna

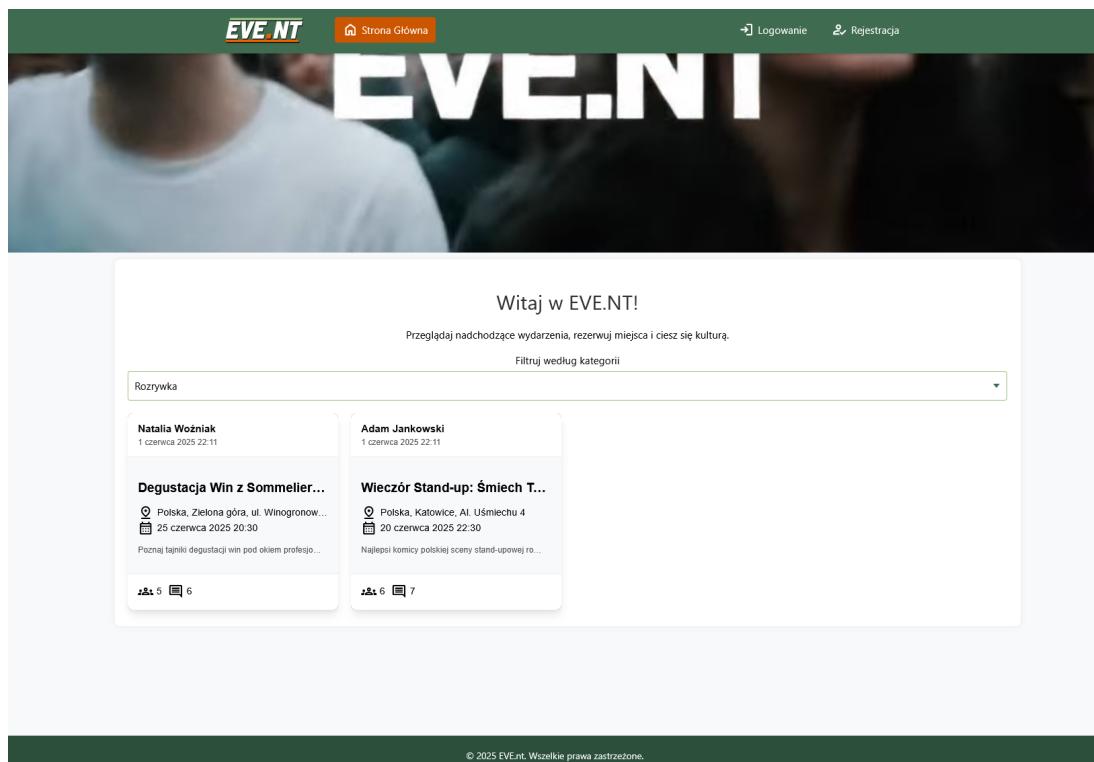
Strona główna (Rys. 1) wita użytkownika i prezentuje listę nadchodzących wydarzeń. Umożliwia szybkie przeglądanie dostępnych opcji oraz nawigację do szczegółów poszczególnych wydarzeń. W górnej części znajduje się pasek nawigacyjny z opcjami logowania, rejestracji, oraz dla zalogowanych użytkowników, dostępem do profilu.



Rys. 1: Widok strony głównej aplikacji.



Rys. 2: Widok strony głównej z przewijaniem listy wydarzeń.



Rys. 3: Strona główna po zastosowaniu filtrów.

5.7.2. Strona Szczegółów Wydarzenia

Po wybraniu konkretnego wydarzenia, użytkownik przechodzi do strony szczegółów (Rys. 5). Prezentowane są tu informacje o wydarzeniu, takie jak opis, data, lokalizacja i organizator. Zalogowani użytkownicy mogą zarezerwować miejsce, a organizatorzy – edytować lub usunąć wydarzenie.

The screenshot shows the 'EVE NT' website interface. At the top, there's a dark green header bar with the 'EVE NT' logo on the left, and 'Strona Główna' (Main Page), 'Logowanie' (Login), and 'Rejestracja' (Registration) buttons on the right. Below the header is a white content area for an event detail page. The title 'Warsztaty Tańców Ludowych: Mazurek i Oberek' is centered at the top. Below the title, there's a light gray footer bar containing the text 'Organizator: Aleksandra Krawczyk' and 'Opublikowano: 1 czerwca 2025 22:11'. The main content is divided into two columns: 'Opis Wydarzenia' (Event Description) on the left and 'Szczegóły' (Details) on the right. The 'Opis Wydarzenia' section contains a short text about learning traditional Polish dances. The 'Szczegóły' section includes fields for 'Data i Godzina' (Date and Time) set to '3 czerwca 2025 13:00', 'Lokalizacja' (Location) set to 'Polska, Zakopane, ul. Folklorystyczna 2', and 'Liczba rezerwacji' (Number of reservations) set to '10/25 (Pozostało: 15)'. A note at the bottom of this column says 'Zaloguj się, aby zarezerwować miejsce.' (Log in to reserve a place.). Below these sections is a 'Komentarze' (Comments) section. It shows three comments from users: 'Tomasz Zieliński' with the message 'O, to coś dla mnie! 🙌', 'Monika Dąbrowska' with 'No to super! Do zobaczenia! 🎉', and 'Agnieszka Król' (comment not visible). All comments are timestamped as '1 czerwca 2025 20:13'.

Rys. 4: Widok strony szczegółów wydarzenia.

Rys. 5: Widok strony szczegółów wydarzenia dla zalogowanego użytkownika.

Po kliknięciu **Zarezerwuj Miejsce**, strona jest odpowiednio aktualizowana i użytkownik dostaje powiadomienie.

Rys. 6: Rezerwacja miejsca.

Sekcja komentarzy U dołu strony szczegółów znajduje się sekcja komentarzy. Każdy zalogowany użytkownik może:

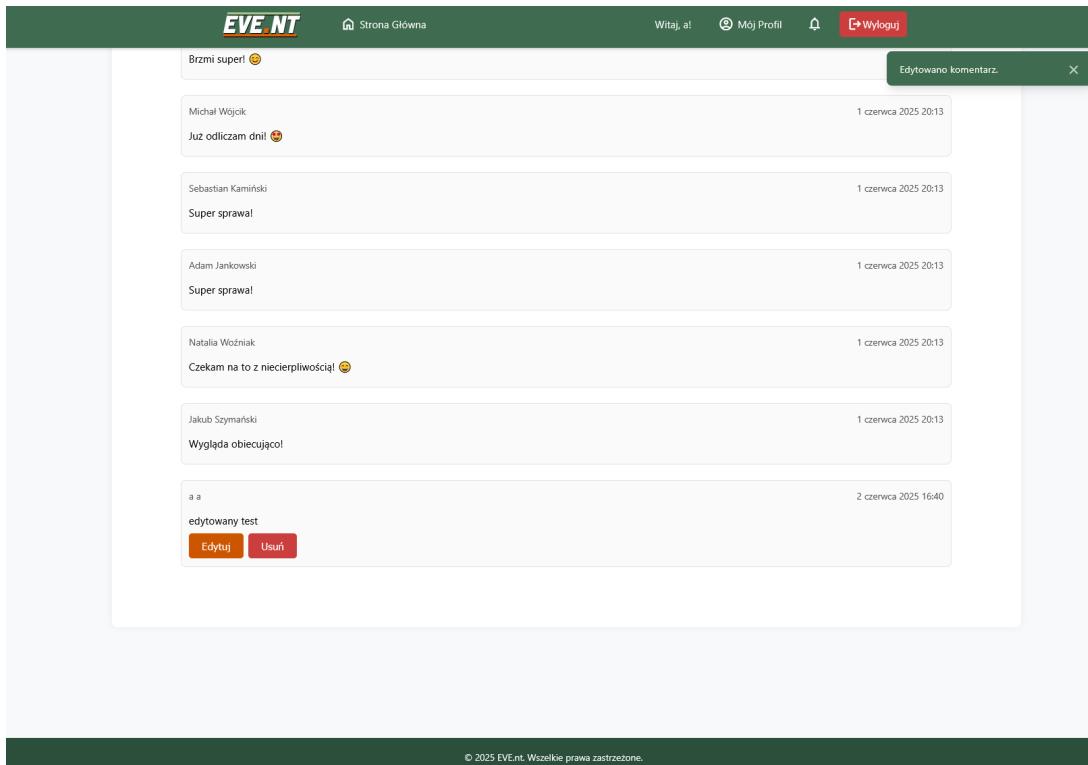
- dodać nowy komentarz,
- edytować lub usunąć swój własny komentarz,
- przeglądać historię dyskusji z innymi uczestnikami.

Komentarze są uporządkowane chronologicznie, a operacje edycji i usuwania są dostępne za pomocą ikon przy każdym wpisie.

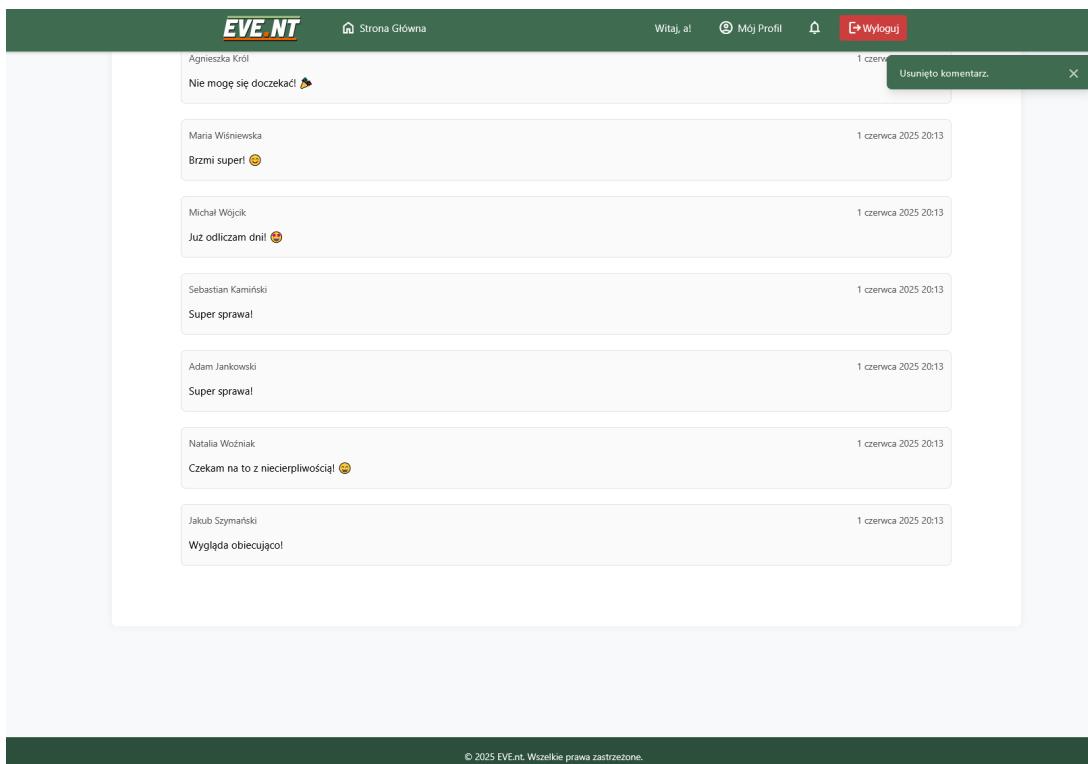
The screenshot shows a list of comments in a dark-themed interface. At the top right, there are user profile icons for 'Witaj, a!', 'Mój Profil', and 'Wyloguj'. Below the header, a message says 'No to super! Do zobaczenia! 🎉'. A green button on the right says 'Dodano komentarz.' with a small 'X' icon. The comments are listed in reverse chronological order:

- Agnieszka Król: Nie mogę się doczekać! 🎉 (1 czerwca 2025 20:13)
- Maria Wiśniewska: Brzmi super! 😊 (1 czerwca 2025 20:13)
- Michał Wójcik: Już odliczam dni! 😊 (1 czerwca 2025 20:13)
- Sebastian Kamiński: Super sprawal! (1 czerwca 2025 20:13)
- Adam Jankowski: Super sprawal! (1 czerwca 2025 20:13)
- Natalia Woźniak: Czekam na to z niecierpliwością! 😊 (1 czerwca 2025 20:13)
- Jakub Szymański: Wygląda obiecująco! (1 czerwca 2025 20:13)
- a a: test (2 czerwca 2025 16:40)
Buttons: Edytuj (orange), Usuń (red)

Rys. 7: Widok sekcji komentarzy z dodanym komentarzem.



Rys. 8: Edycja komentarza użytkownika.



Rys. 9: Usunięcie komentarza użytkownika.

5.7.3. Formularze Autentykacji

Aplikacja oferuje czytelne formularze logowania i rejestracji (Rys. 11), z wbudowaną validacją. Błędy są wyświetlane w czasie rzeczywistym, co ułatwia poprawne wypełnienie danych.

Logowanie

Zaloguj się, aby kontynuować.

Nie masz jeszcze konta? [Zarejestruj się](#)

Login (email lub nazwa użytkownika)
Wpisz email lub nazwę użytkownika

Hasło
Wpisz hasło

Zaloguj się

Rejestracja

Dolacz do naszej społeczności i odkrywaj wydarzenia kulturalne.

Masz już konto? [Zaloguj się](#)

Nazwa użytkownika
Wpisz nazwę użytkownika

Imię
Wpisz imię

Nazwisko
Wpisz nazwisko

Data urodzenia (YYYY-MM-DD)
dd / mm / yyyy CALENDAR

Email
Wpisz adres email

Kod kraju
+48

Numer telefonu
123456789

Hasło
Wpisz hasło

Potwierdź hasło
Potwierdź hasło

Zarejestruj

(a) Formularz logowania.

(b) Formularz rejestracji.

Rys. 10: Formularze autentykacji użytkownika.

Rejestracja

Dolacz do naszej społeczności i odkrywaj wydarzenia kulturalne.

Masz już konto? [Zaloguj się](#)

Logowanie

Zaloguj się, aby kontynuować.

Nie masz jeszcze konta? [Zarejestruj się](#)

Niepoprawne dane.

Login (email lub nazwa użytkownika)

asdqwe

Hasło

[Zaloguj się](#)

Nazwa użytkownika

Wpisz nazwę użytkownika

Nazwa użytkownika jest wymagana.

Imię

Wpisz imię

Imię jest wymagane.

Nazwisko

Wpisz nazwisko

Nazwisko jest wymagane.

Data urodzenia (YYYY-MM-DD)

dd / mm / yyyy



Data urodzenia jest wymagana.

Email

ab@c

Niepoprawny format adresu email.

Kod kraju

+48

Numer telefonu

123456789

Hasło

Hasło musi mieć co najmniej 8 znaków.

Potwierdź hasło

Hasła nie są zgodne.

[Zarejestruj](#)

(a) Formularz logowania.

(b) Formularz rejestracji.

Rys. 11: Przykład działania formularzów autentykacji użytkownika.

5.7.4. Panel Użytkownika

Po zalogowaniu użytkownik ma dostęp do panelu zarządzania kontem (Rys. 12). Znajdują się tam sekcje:

- **Moje wydarzenia** – lista wydarzeń utworzonych przez użytkownika oraz nadchodzące wydarzenia, na które zapisano konto,
- **Moje lokalizacje** – lista umożliwiająca dodawanie i usuwanie lokalizacji użytkownika.
- **Moje kategorie** – lista umożliwiająca dodawanie i usuwanie kategorii.

Próba usunięcia lokalizacji lub kategorii, która jest używana przez dowolne wydarzenie, zakończy się niepowodzeniem.

The screenshot shows the user panel interface for the 'EVE NT' platform. At the top, there's a navigation bar with links for 'Strona Główna', 'Witaj, Bartłomiej!', 'Moj Profil', and 'Wyloguj'. Below the navigation, the user's profile information is displayed: 'Bartłomiej Masiak' with a small profile picture placeholder 'a'. There are two buttons: 'Edytuj Profil' and '+ Nowe wydarzenie'. The main content area is divided into three sections: 'Twoje Wydarzenia', 'Twoje Lokalizacje', and 'Twoje Kategorie'.

The 'Twoje Wydarzenia' section shows one event: 'Aleksandra Krawczyk' on '1 czerwca 2025 22:11'. It details an event titled 'Warsztaty Tańców Ludowych: M...', located in 'Polska, Zakopane, ul. Folklorystyczna 2' on '3 czerwca 2025 13:00'. A note says 'Odkryj radość polskich tańców ludowych! Warsztaty d...'. Below the event card, there are statistics: '12' and '9', and a button 'Weźmiesz udział'.

The 'Twoje Lokalizacje' section has a message: 'Nie dodałeś jeszcze żadnych lokalizacji.' and a button 'Dodaj Lokalizację'.

The 'Twoje Kategorie' section lists categories: 'Nazwa' (Folklor, Edukacja, Sztuka, Klasyka, Fotografia) and 'Akcje' (Usuń buttons for each category). Each category row has a 'Usuń' button.

Rys. 12: Widok panelu użytkownika.

The screenshot shows the EVE.NT platform interface. At the top, there is a navigation bar with links for 'Strona Główna' (Main Page), 'Witaj, Bartłomiej!', 'Mój Profil' (My Profile), and 'Wyloguj' (Logout). A red error message box is displayed, stating: 'Błąd usuwania kategorii. Może być wykorzystywana przez obecnie istniejące wydarzenie.' (Category deletion error. It may be used by existing events.)

The main content area is titled 'Twoje Wydarzenia' (Your Events) and shows a single event card for 'Warsztaty Tańców Ludowych: M...'. The event details include: Aleksandra Krawczyk, 1 czerwca 2025 22:11, Polska, Zakopane, ul. Folklorystyczna 2, 3 czerwca 2025 13:00, and a description: 'Odkryj rodzaj polskich tańców ludowych! Warsztaty d...'. Below the event card, there are two buttons: 'Edytuj Wydarzenie' (Edit Event) and '+ Nowe wydarzenie' (New Event).

Below the events section is a 'Twoje Lokalizacje' (Your Locations) section, which currently displays the message: 'Nie dodaleś jeszcze żadnych lokalizacji.' (You haven't added any locations yet.). A button '+ Dodaj Lokalizację' (Add Location) is available.

The final section shown is 'Twoje Kategorie' (Your Categories), listing the following categories with 'Akcje' (Actions) buttons:

Nazwa	Akcje
Folklor	
Edukacja	
Sztuka	
Klasyka	
Fotografia	

Rys. 13: Próba usunięcia używanej kategorii 'Folklor'.

5.7.5. Formularz Tworzenia i Edycji Wydarzenia

Tworzenie lub edycja wydarzenia odbywa się za pomocą rozbudowanego formularza. Użytkownik może po- dać:

- nazwę, opis, datę i godzinę,
- lokalizację i kategorie,
- limit miejsc

Pola są walidowane, a interfejs umożliwia szybkie wprowadzanie zmian w edytowanych wydarzeniach.

Rys. 14: Widok formularza tworzenia wydarzenia.

Rys. 15: Widok po stworzeniu wydarzenia.

Rys. 16: Formularz edycji istniejącego wydarzenia.

Rys. 17: Widok po edycji wydarzenia.

6. Aplikacja Mobilna

Aplikacja mobilna EVE.NT została stworzona z myślą o użytkownikach systemu Android, zapewniając im wygodny dostęp do portalu wydarzeń kulturalnych. Umożliwia przeglądanie, rezerwowanie, komentowanie oraz zarządzanie wydarzeniami i własnym profilem bezpośrednio z urządzenia mobilnego.

6.1. Wprowadzenie i Technologie

Celem aplikacji mobilnej jest dostarczenie pełnej funkcjonalności portalu EVE.NT w natywnym środowisku Android. Skupiliśmy się na intuicyjnym interfejsie użytkownika oraz płynnym działaniu, wykorzystując standardowe komponenty i wzorce projektowe platformy Android.

Do budowy aplikacji mobilnej wykorzystaliśmy następujące technologie i narzędzia:

- **Język programowania:** Java.
- **Platforma:** Android SDK.
- **Architektura:** Aplikacja oparta na Fragmentach, z wykorzystaniem wzorca Repository.
- **Nawigacja:** Android Navigation Component do zarządzania przejściami między ekranami, w połączeniu z BottomNavigationView dla głównej nawigacji.
- **Interfejs użytkownika:** Layouty definiowane w XML, z wykorzystaniem komponentów Material Design. Style aplikacji wspierają zarówno tryb jasny, jak i ciemny systemu operacyjnego.
- **Wyświetlanie list:** RecyclerView do efektywnego prezentowania dynamicznych list danych.
- **Komunikacja sieciowa:** Wbudowana klasa HttpURLConnection opakowana w AsyncTask do realizacji żądań HTTP do API backendu.
- **Obsługa danych:** Parsowanie odpowiedzi JSON z API za pomocą biblioteki org.json.
- **Zarządzanie tokenami:** Dedykowana klasa TokenManager do przechowywania i dostarczania tokenów autoryzacyjnych.
- **Konfiguracja bezpieczeństwa:** Pliki network_security_config.xml, data_extraction_rules.xml oraz backup_rules.xml definiujące zasady bezpieczeństwa.

6.2. Architektura Aplikacji

Aplikacja została zaprojektowana z myślą o modularności i łatwości rozbudowy, opierając się na architekturze opartej na fragmentach i komponencie nawigacji.

6.2.1. Struktura Projektu

Główne komponenty aplikacji zgrupowane są w standardowych dla Androida pakietach:

- **com.example.event.data:** Zawiera klasy modeli danych, repozytoria oraz klasy pomocnicze. Model Event.java to klasa agregująca wszystkie dane dotyczące wydarzenia, takie jak ID, tytuł, daty, dane lokalizacji, organizatora, status rezerwacji, liczbę uczestników i komentarzy.
- **com.example.event.ui:** Gromadzi Fragmenty i Adaptery dla interfejsu użytkownika.
- **res/layout:** Pliki XML definiujące wygląd ekranów i elementów list.
- **res/navigation:** Graf nawigacji mobile_navigation.xml.

6.2.2. Nawigacja

Nawigacja w aplikacji jest realizowana przy użyciu **Android Navigation Component**. Główna nawigacja odbywa się za pomocą `BottomNavigationView`, przełączając między `HomeFragment`, `NotificationsFragment` i `UserFragment`. Przepływ między wszystkimi ekranami zdefiniowany jest w grafie nawigacji `mobile_navigation.xml`.

6.2.3. Zarządzanie Danymi i Stanem

Stan aplikacji, w tym dane zalogowanego użytkownika i tokeny, zarządzany jest głównie przez `LoginRepository` oraz `TokenManager`. Modele danych reprezentują obiekty pobierane z API. Operacje w tle i aktualizacje UI wykorzystują `AsyncTask` bezpośrednio w Fragmentach.

6.3. Główne Komponenty i Widoki

Kluczowe funkcjonalności są zaimplementowane jako osobne Fragmenty.

6.3.1. Kluczowe Fragmenty

- **HomeFragment:** Główny ekran z listą wydarzeń, przyciskami FAB do dodawania wydarzeń i filtrowania.
- **EventDetailFragment:** Szczegóły wydarzenia, komentarze, opcje edycji/usunięcia dla organizatora.
- **AddEventFragment:** Formularz tworzenia wydarzeń. Implementacja obejmuje inicjalizację pól formularza, walidację danych wejściowych oraz komunikację z API w celu utworzenia nowej lokalizacji i samego wydarzenia.
- **EditEventFragment:** Formularz edycji wydarzenia, wstępnie wypełniany danymi.
- **LoginFragment i RegisterFragment:** Ekrany autentykacji.
- **UserFragment:** Profil użytkownika z listami jego wydarzeń, rezerwacji oraz opcjami edycji profilu i wylogowania.
- **FilterFragment:** Panel filtrowania wydarzeń.
- **NotificationsFragment:** Lista powiadomień.

6.3.2. Elementy List (RecyclerView Adapters)

- **EventAdapter:** Wykorzystywany do wyświetlania kart wydarzeń w `RecyclerView`. Każda karta, zdefiniowana w `item_event.xml`, prezentuje kluczowe informacje o wydarzeniu, takie jak autor, tytuł, daty, lokalizacja, opis, liczba uczestników i komentarzy. Umożliwia również interakcję poprzez przyciski dołączenia/rezygnacji oraz rozwijania sekcji komentarzy.
- **CommentAdapter:** Służy do wyświetlania listy komentarzy. Umożliwia autorowi komentarza jego edycję lub usunięcie.
- **NotificationAdapter:** Prezentuje listę powiadomień dla użytkownika.

6.4. Interakcja z Backendem

Komunikacja z API backendu odbywa się za pomocą `HttpURLConnection` wewnętrz `AsyncTask`. Dane wymieniane są w formacie JSON. Bazowy adres API jest przechowywany w klasie `ApiConfig`. Tokeny JWT są zarządzane przez `TokenManager` i dołączane do chronionych żądań.

6.5. Zarządzanie Stanem Użytkownika

Autentykacja odbywa się przez `LoginFragment` i `RegisterFragment`, z wykorzystaniem `LoginRepository`. Tokeny i dane użytkownika są bezpiecznie przechowywane. Dostęp do chronionych sekcji aplikacji jest warunkowany statusem zalogowania.

6.6. System Filtrowania i Sortowania Wydarzeń

FilterFragment pozwala na wybór kategorii (za pomocą komponentów Chip) oraz kryteriów sortowania. Wybrane filtry są przekazywane z powrotem do HomeFragment, który odświeża listę wydarzeń. Aktywne filtry są sygnalizowane zmianą wyglądu przycisku FAB oraz plakietką na przycisku resetowania w panelu filtrów.

6.7. System Komentarzy

Użytkownicy mogą dodawać, przeglądać, edytować i usuwać własne komentarze pod wydarzeniami. Operacje te komunikują się z odpowiednimi endpointami API.

6.8. System Powiadomień

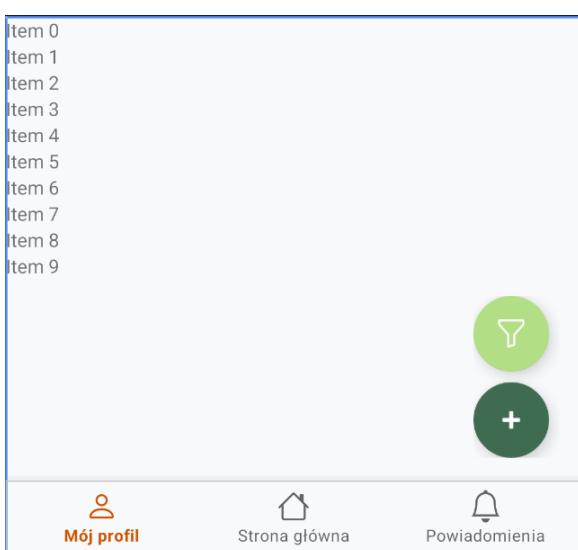
Sekcja powiadomień wyświetla listę alertów z serwera.

- **Sygnalizacja Nowych Powiadomień:** Ikona zakładki "Powiadomienia" w dolnym pasku nawigacyjnym wyświetla badge z liczbą nieprzeczytanych powiadomień.
- **Oznaczanie jako Przeczytane:** Po wejściu użytkownika do zakładki powiadomień, aplikacja wysyła żądanie do API w celu oznaczenia powiadomień jako przeczytane. Wskaźniki nieprzeczytanych wiadomości na liście znikają, a badge w nawigacji jest aktualizowany.

Aplikacja mobilna nie używa natywnych powiadomień push systemu Android.

6.9. Struktura Aplikacji i Nawigacja

Podstawowa struktura aplikacji oraz nawigacja:



(a) Struktura pliku `activity_main.xml`.

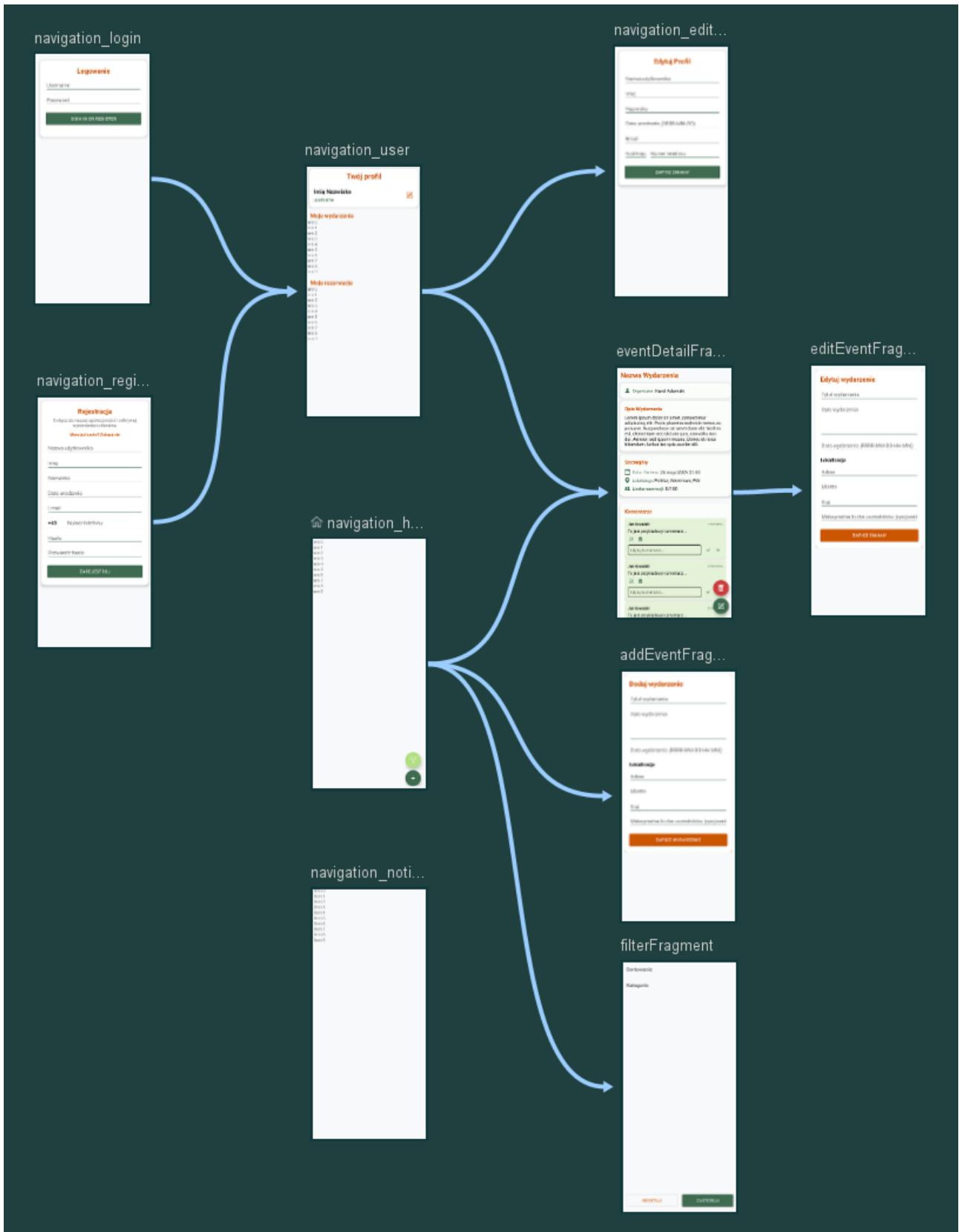
```
<item
    android:id="@+id/navigation_user"
    android:icon="@drawable/ic_person"
    android:title="Mój profil"
    android:menuCategory="container" />

<item
    android:id="@+id/navigation_home"
    android:icon="@drawable/ic_house"
    android:title="Strona główna"
    android:menuCategory="container" />

<item
    android:id="@+id/navigation_notifications"
    android:icon="@drawable/ic_bell"
    android:title="Powiadomienia"
    android:menuCategory="container" />
```

(b) Definicja menu `bottom_nav_menu.xml` (kod XML).

Rys. 18: Kluczowe pliki XML definiujące strukturę i nawigację (widok z IDE).



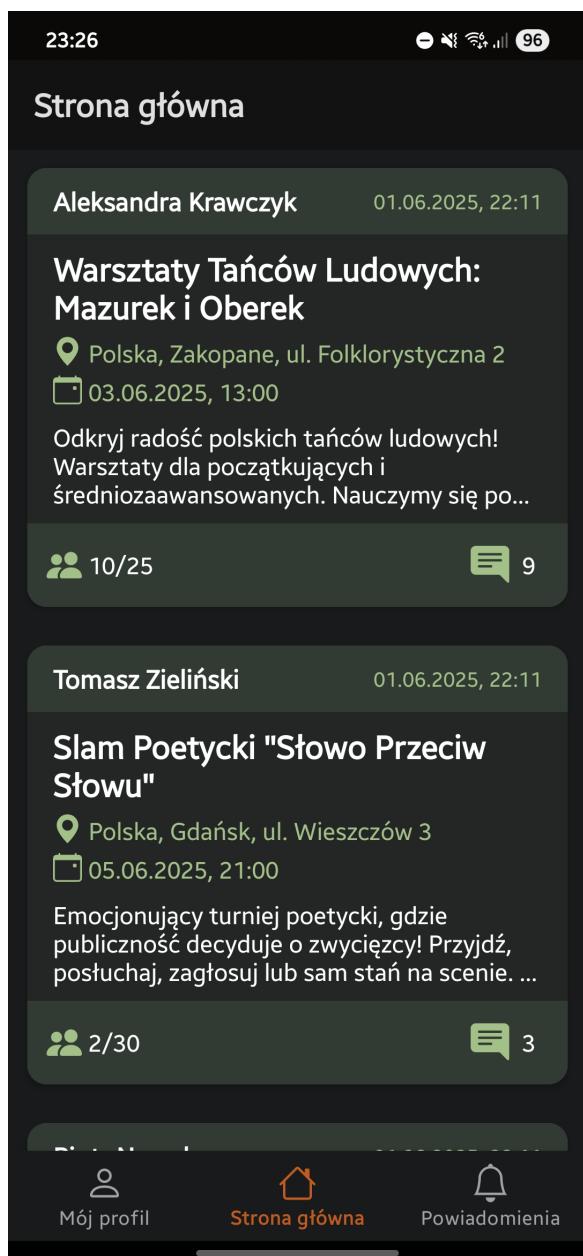
Rys. 19: Graf nawigacji (`mobile_navigation.xml`).

6.10. Efekty działania aplikacji

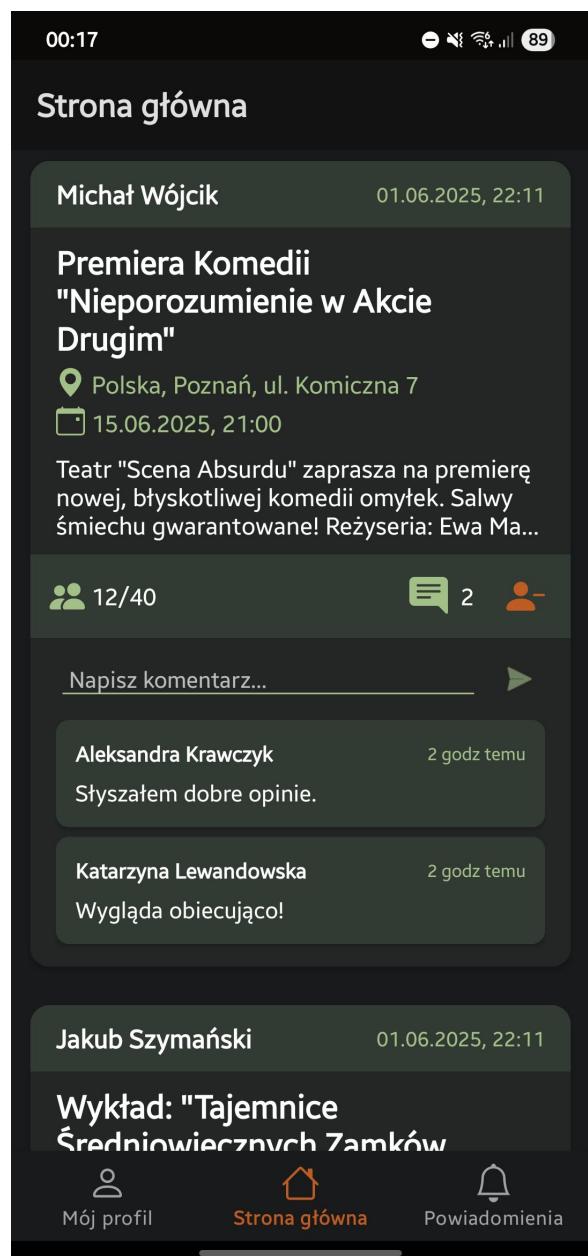
Poniżej zaprezentowano zrzuty ekranu z telefonu, ilustrujące kluczowe funkcjonalności aplikacji mobilnej EVE.NT, a także ważne elementy strukturalne, takie jak layouty i graf nawigacji.

6.10.1. Ekran Główny i Lista Wydarzeń

Ekran główny (HomeFragment) prezentuje listę dostępnych wydarzeń. Wygląd i dostępne interakcje na liście wydarzeń różnią się w zależności od statusu zalogowania użytkownika. Każdy element listy zawiera kluczowe informacje o wydarzeniu oraz przyciski umożliwiające interakcję, takie jak dołączenie do wydarzenia czy rozwinięcie sekcji komentarzy.



(a) Widok użytkownika niezalogowanego.



(b) Widok użytkownika zalogowanego.

Rys. 20: Lista wydarzeń na ekranie głównym.

6.10.2. Szczegóły Wydarzenia i Komentarze

Ekran szczegółów wydarzenia (Rys. 21a) wyświetla pełne informacje oraz sekcję komentarzy (Rys. 21b).

The image consists of two side-by-side screenshots of a mobile application interface.

Screenshot (a): Szczegóły wydarzenia

- Top Bar:** Shows the time (23:29), battery level (96%), and signal strength.
- Title:** "Szczegóły wydarzenia" with a back arrow.
- Event Title:** "Wieczór Jazzowy z Trio Nowoczesnym".
- Organizer:** Anna Kowalska.
- Opis Wydarzenia:** Text describing the event as a jazz evening with improvisations and modern standards by the Trio Nowoczesny.
- Szczegóły:** Includes date (07.06.2025, 22:00), location (Polska, Warszawa, ul. Jazzowa 5), and number of reservations (10/40).
- Komentarze:** A section with a "Dodaj komentarz..." input field and a green "Zapisz" button. It also shows a trash icon and a notification badge with the number 5.
- Bottom Navigation:** Icons for profile (Anna), main page, and notifications (5).

Screenshot (b): Sekcja komentarzy

- Top Bar:** Shows the time (23:30), battery level (96%), and signal strength.
- Title:** "Szczegóły wydarzenia" with a back arrow.
- Section Header:** "Komentarze".
- Input Field:** "Dodaj komentarz..." with a right-pointing arrow.
- Comments List:** A vertical list of user posts:
 - Anna Kowalska: "Może wpadnę..." (1 godz temu)
 - Natalia Woźniak: "Brzmi super!" (1 godz temu)
 - Katarzyna Lewandowska: "Idealnie!" (1 godz temu)
 - Grzegorz Adamski: "Brzmi super!" (1 godz temu)
 - Piotr Nowak: "Zapowiada się świetna zabawa!" (1 godz temu)
 - Maria Wiśniewska: "Mam nadzieję, że uda mi się wpaść." (1 godz temu)
 - Adam Jankowski: "..." (1 godz temu)
- Bottom Navigation:** Icons for profile (Anna), main page, and notifications (5).

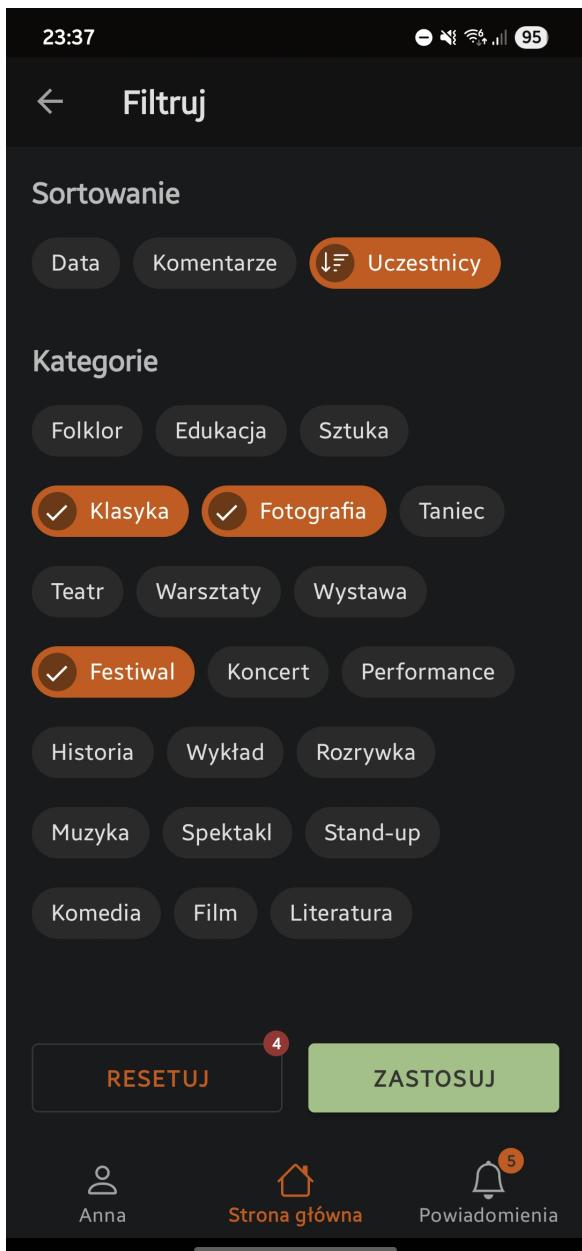
(a) Szczegóły wydarzenia.

(b) Sekcja komentarzy.

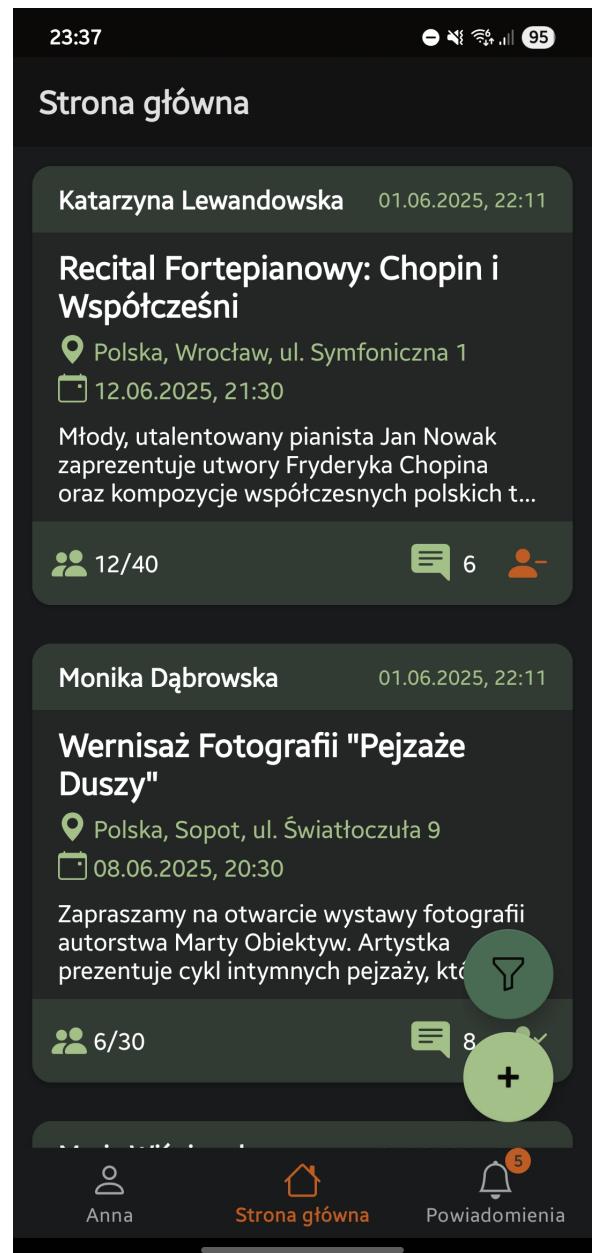
Rys. 21: Widok szczegółów wydarzenia i sekcji komentarzy.

6.10.3. Filtrowanie i Sortowanie

Aplikacja umożliwia zaawansowane filtrowanie wydarzeń według kategorii oraz ich sortowanie według różnych kryteriów. Panel filtrowania pozwala użytkownikowi na precyzyjne dostosowanie wyświetlanej listy wydarzeń.



(a) Panel filtrowania z opcjami wyboru kategorii oraz kryteriów sortowania.



(b) Widok listy wydarzeń po zastosowaniu filtrów kategorii i wybranego sortowania.

6.10.4. Dodawanie i Edycja Wydarzenia

Aplikacja umożliwia zalogowanym użytkownikom tworzenie nowych wydarzeń oraz edycję tych, których są organizatorami. Formularze zawierają wszystkie niezbędne pola do zdefiniowania wydarzenia.

23:39

← Dodaj wydarzenie

Dodaj wydarzenie

Tytuł wydarzenia

Opis wydarzenia

Data wydarzenia (RRRR-MM-DD HH:MM)

Lokalizacja

Adres

Miasto

Kraj

Maksymalna liczba uczestników (op.)

ZAPISZ WYDARZENIE

Anna

Strona główna

Powiadomienia (5)

(c) Formularz dodawania nowego wydarzenia.

23:43

← Edytuj wydarzenie

Edytuj wydarzenie

Wieczór Jazzowy z Trio Nowoczesny

Zapraszamy na niezapomniany wieczór pełen improwizacji i jazzowych standardów w wykonaniu utalentowanego Trio Nowoczesnego. Ich muzyka to...

2025-06-07 22:00

Lokalizacja

ul. Jazzowa 5

Warszawa

Polska

40

ZAPISZ ZMIANY

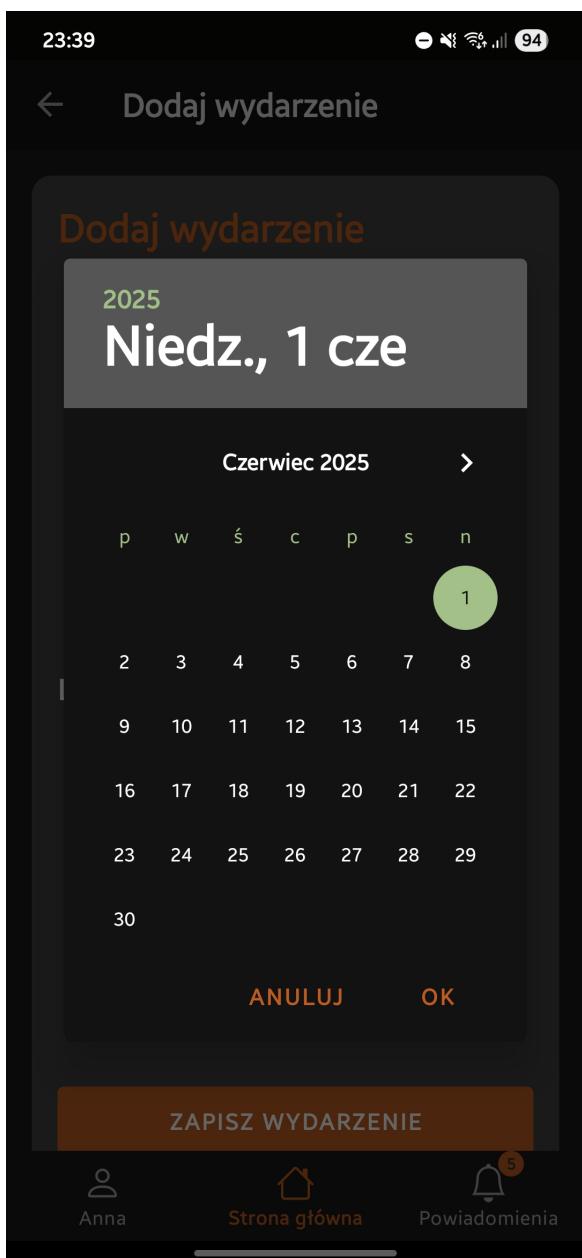
Anna

Strona główna

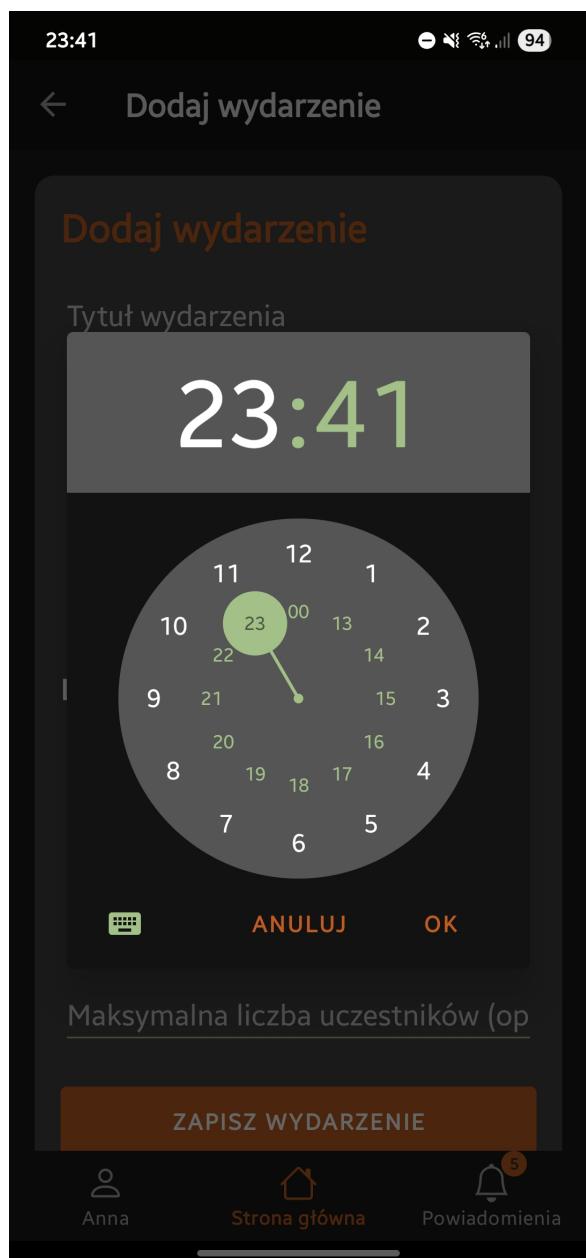
Powiadomienia (5)

(d) Formularz edycji istniejącego wydarzenia.

Rys. 22: Formularze zarządzania wydarzeniami: dodawanie i edycja.



(a) Widok selektora daty

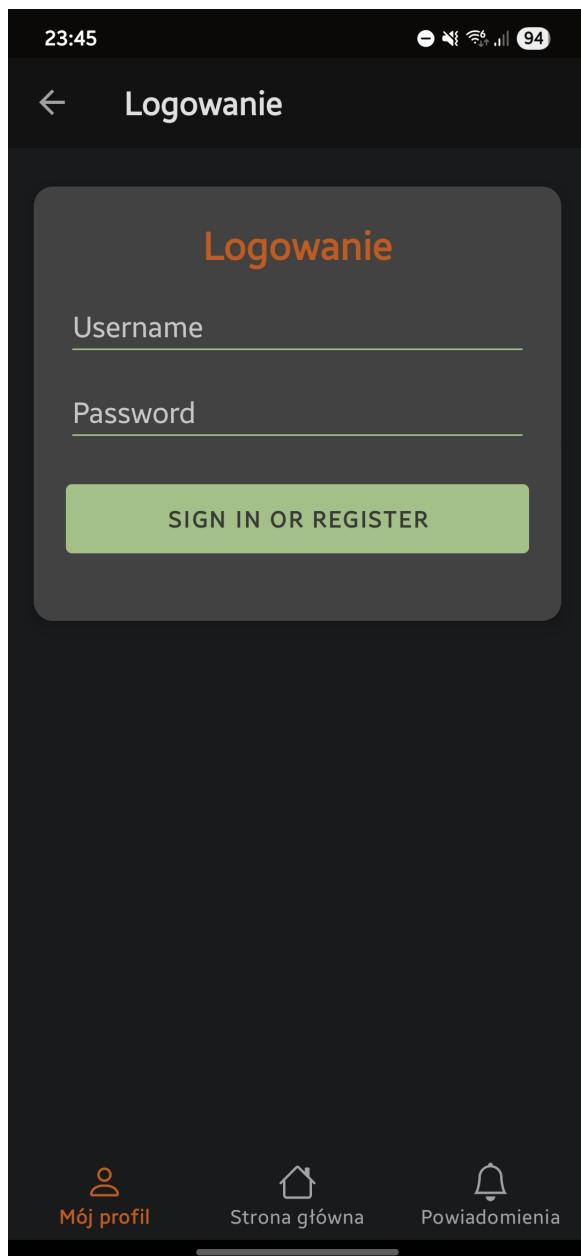


(b) Widok selektora godziny

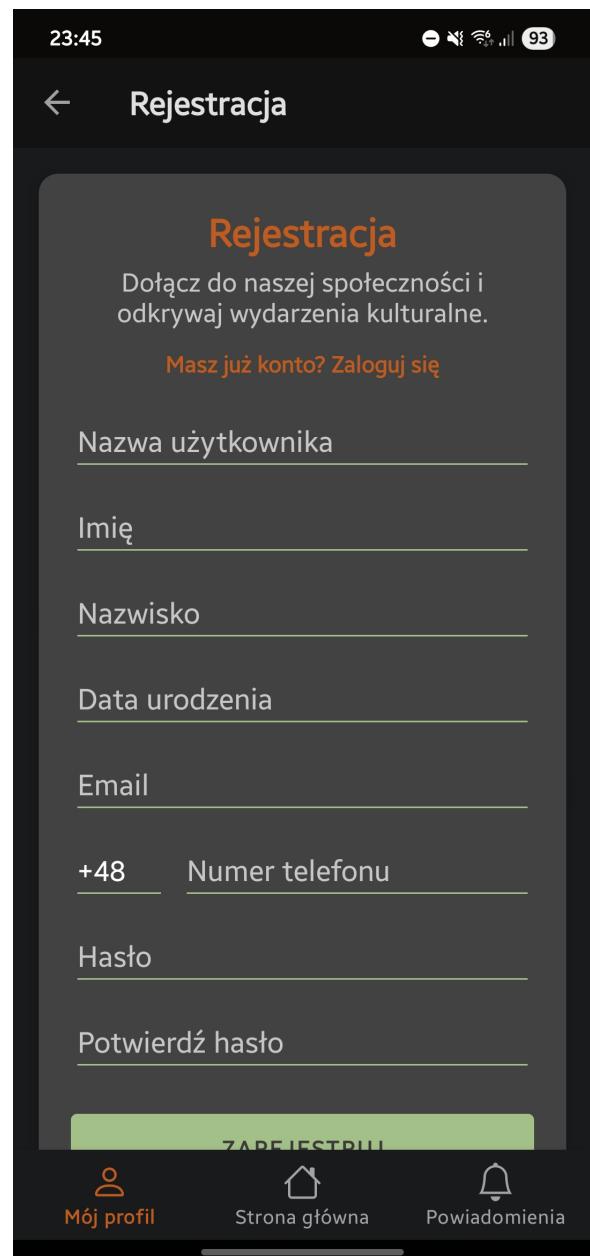
Rys. 23: Wybór daty i godziny wydarzenia.

6.10.5. Autentykacja Użytkownika

Ekrany logowania (Rys. 24a) i rejestracji (Rys. 24b).



(a) Ekran logowania.

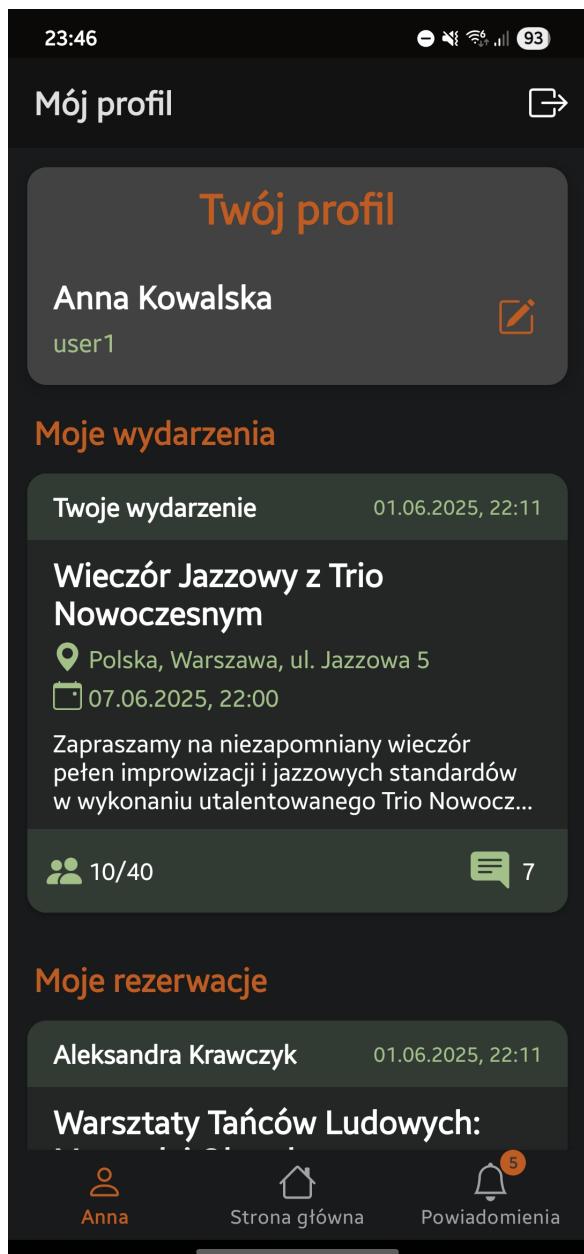


(b) Ekran rejestracji.

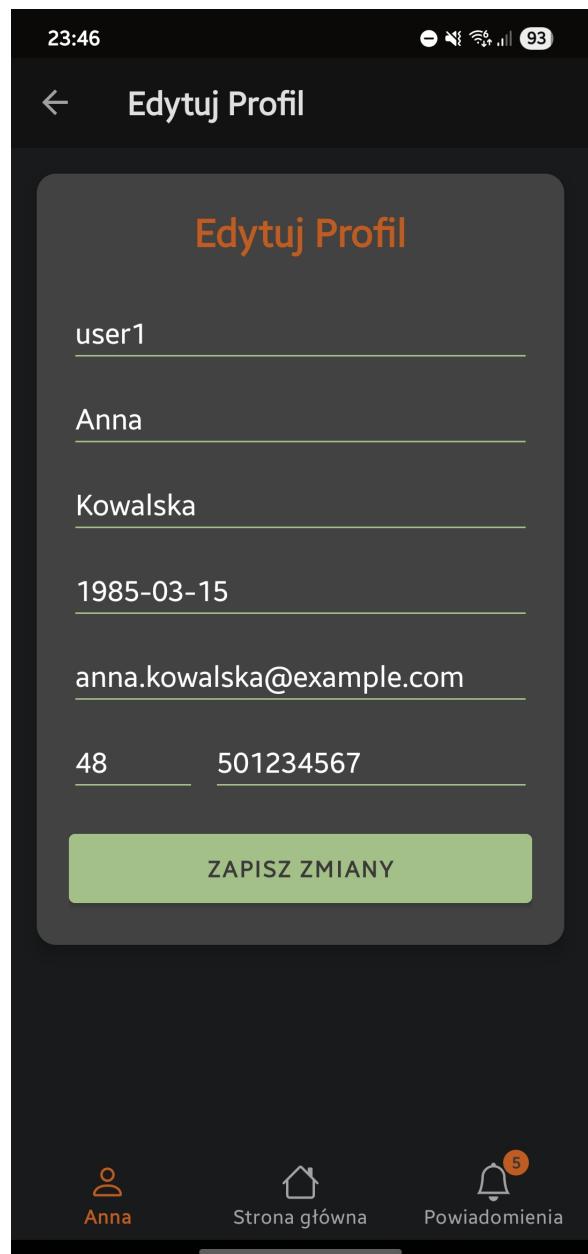
Rys. 24: Ekrany autentykacji użytkownika.

6.10.6. Profil Użytkownika i Edycja Profilu

Profil zalogowanego użytkownika (Rys. 25a) oraz formularz edycji danych (Rys. 25b).



(a) Widok profilu zalogowanego użytkownika.



(b) Formularz edycji profilu użytkownika.

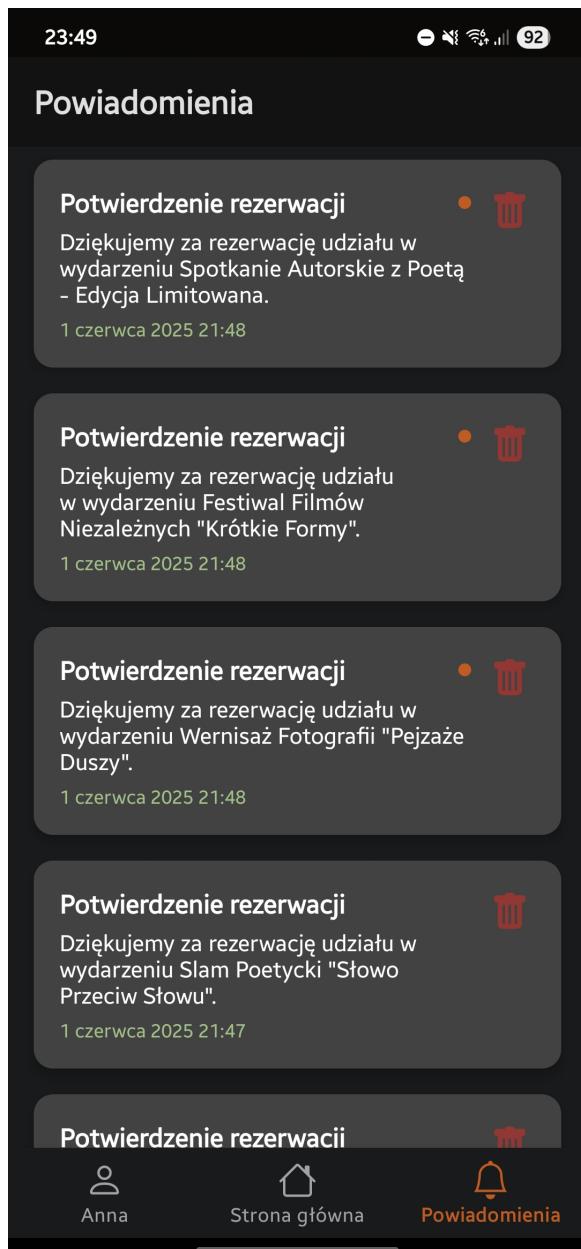
Rys. 25: Ekran profilu użytkownika i edycji profilu.

6.10.7. System Powiadomień - Działanie

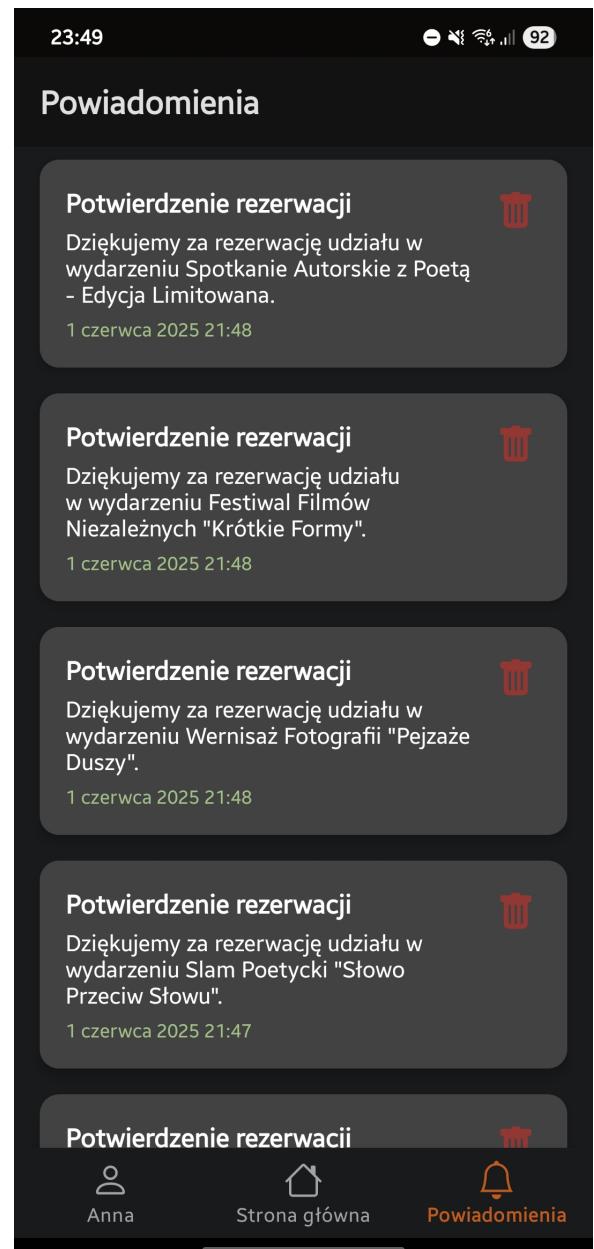
System powiadomień informuje użytkownika o nowych zdarzeniach.



Rys. 26: Dolny pasek nawigacyjny z badge'em powiadomień.



(a) Lista powiadomień.

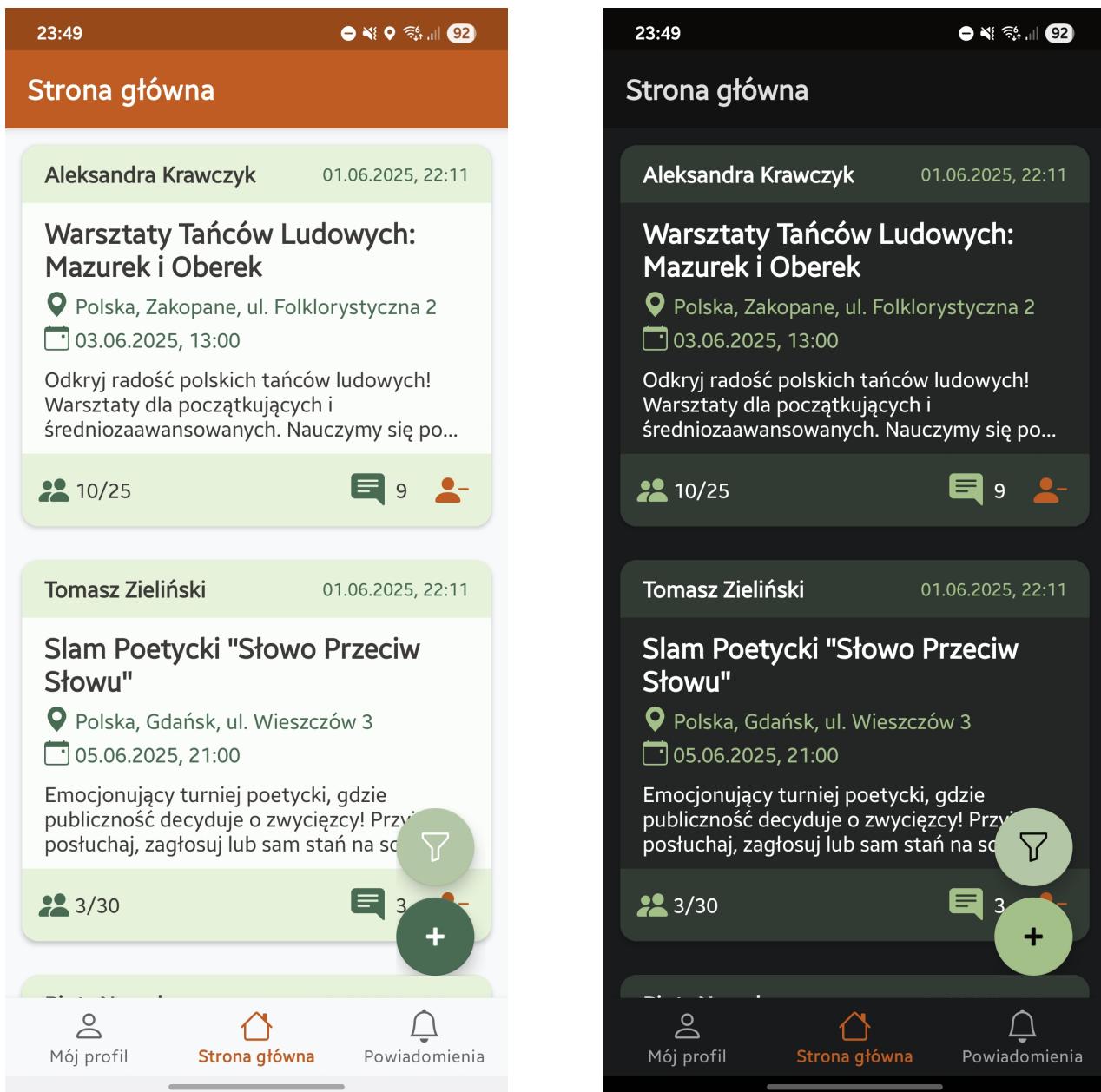


(b) Lista powiadomień po oznaczeniu jako przeczytane.

Rys. 27: Element powiadomienia i widok listy po interakcji.

6.10.8. Obsługa Trybu Jasnego i Ciemnego

Aplikacja dostosowuje swój wygląd do systemowego ustawienia trybu jasnego lub ciemnego, zapewniając komfort użytkowania w różnych warunkach oświetleniowych (Rys. 28).



(a) Przykład ekranu w trybie jasnym.

(b) Ten sam ekran w trybie ciemnym.

Rys. 28: Demonstracja obsługi trybu jasnego i ciemnego na przykładzie ekranu głównego.

7. Opis aplikacji back-end

7.1. Architektura Aplikacji

Aplikacja backendowa systemu EVE.NT została zaimplementowana z wykorzystaniem frameworka Flask i zaprojektowana jako modularne REST API. Celem projektu było stworzenie skalowalnej, bezpiecznej i łatwej w rozbudowie architektury umożliwiającej komunikację zarówno z aplikacją webową, jak i mobilną.

7.1.1. Struktura Projektu

Struktura katalogów backendu prezentuje się następująco:

- **controllers/** – zawiera blueprinty i endpointy REST API dla poszczególnych zasobów aplikacji, takich jak autentykacja (`auth.py`), użytkownicy, wydarzenia, komentarze, rezerwacje, lokalizacje i kategorie.
- **models/** – zawiera definicje modeli danych w oparciu o SQLAlchemy. Znajdują się tu klasy reprezentujące encje bazy danych, m.in. `User`, `Event`, `Reservation`, `Comment`, `Location` i `Category`.
- **services/** – moduły zawierające logikę biznesową aplikacji, np. tworzenie wydarzeń, rejestracja użytkowników, obsługa rezerwacji czy komentowania.
- **utils/** – zbiór funkcji pomocniczych, takich jak generowanie i weryfikacja tokenów JWT, obsługa błędów, walidacja danych wejściowych.
- **extensions.py** – plik odpowiedzialny za inicjalizację globalnych rozszerzeń Flask: SQLAlchemy (`db`), Flask-Migrate (`migrate`), Flask-Mail (`mail`).
- **config.py** – definicja klasy `Config`, która ładuje zmienne środowiskowe z pliku `.env`, takie jak klucze JWT, dane dostępowe do bazy danych PostgreSQL, ustawienia maila i parametr DEBUG.
- **__init__.py** – funkcja `create_app()` tworzy i konfigurujeinstancję aplikacji Flask: rejestruje blueprinty, inicjalizuje rozszerzenia i logowanie oraz tworzy strukturę bazy danych.

7.1.2. Baza Danych

Backend korzysta z bazy danych PostgreSQL, połączonej za pomocą SQLAlchemy ORM. Modele danych odzwierciedlają strukturę logiczną aplikacji:

- `User` – dane użytkownika, takie jak login, e-mail, imię, nazwisko, hasło (hashowane), numer telefonu i rola.
- `Event` – informacje o wydarzeniu, m.in. tytuł, opis, daty, lokalizacja, organizator, liczba dostępnych miejsc.
- `Location` – dane dotyczące lokalizacji wydarzeń.
- `Category` – kategorie wydarzeń.
- `Reservation` – relacja użytkownika z wydarzeniem (rezerwacja miejsca).
- `Comment` – komentarze użytkowników przypisane do wydarzeń.

Relacje pomiędzy modelami uwzględniają m.in. związki wiele-do-wielu pomiędzy wydarzeniami a kategoriami.

7.1.3. Autoryzacja i Bezpieczeństwo

Autentykacja użytkownika opiera się na tokenach JWT. System wykorzystuje tokeny dostępowe (Access Token) oraz odświeżające (Refresh Token) o czasie życia konfigurowanym w pliku `.env`. Wszystkie hasła są przechowywane w postaci zahashowanej (np. z użyciem bcrypt), a dostęp do chronionych endpointów wymaga obecności tokenu JWT w nagłówku `Authorization: Bearer <token>`.

7.1.4. Endpointy API

Backend aplikacji udostępnia szereg endpointów REST API zorganizowanych w moduły logiczne. Poniżej przedstawiono pełną listę endpointów:

- **Autoryzacja (/auth)**
 - `POST /auth/login` – logowanie użytkownika,
 - `POST /auth/register` – rejestracja nowego użytkownika,
 - `POST /auth/refresh` – odświeżenie tokenu JWT.

- **Użytkownicy (/users)**
 - GET /users/ – pobranie listy wszystkich użytkowników (autoryzacja wymagana),
 - GET /users/<user_id> – pobranie danych konkretnego użytkownika (autoryzacja wymagana),
 - GET /users/name/<user_id> – pobranie imienia i nazwiska użytkownika,
 - POST /users/ – utworzenie użytkownika,
 - PUT /users/<user_id> – edycja danych użytkownika (autoryzacja wymagana),
 - DELETE /users/<user_id> – usunięcie użytkownika (autoryzacja wymagana).
- **Wydarzenia (/api/events)**
 - GET /api/events/ – pobranie wszystkich wydarzeń,
 - GET /api/events/user/<user_id> – wydarzenia danego użytkownika,
 - GET /api/events/category/<category_id> – wydarzenia z wybranej kategorii,
 - GET /api/events/<event_id> – szczegóły wybranego wydarzenia,
 - POST /api/events/ – utworzenie wydarzenia (autoryzacja wymagana),
 - PUT /api/events/<event_id> – edycja wydarzenia (autoryzacja wymagana),
 - DELETE /api/events/<event_id> – usunięcie wydarzenia (autoryzacja wymagana).
- **Rezerwacje (/api/reservations)**
 - GET /api/reservations/ – lista rezerwacji,
 - POST /api/reservations/ – utworzenie nowej rezerwacji (autoryzacja wymagana),
 - DELETE /api/reservations/<reservation_id> – usunięcie rezerwacji (autoryzacja wymagana),
 - GET /api/reservations/confirm/<token> – potwierdzenie rezerwacji przez link z tokenem.
- **Komentarze (/api/comments)**
 - POST /api/comments/ – dodanie komentarza (autoryzacja wymagana),
 - GET /api/comments/<event_id> – komentarze do danego wydarzenia,
 - PUT /api/comments/<comment_id> – edycja komentarza (autoryzacja wymagana),
 - DELETE /api/comments/<comment_id> – usunięcie komentarza (autoryzacja wymagana).
- **Kategorie (/api/category)**
 - GET /api/category/ – lista kategorii,
 - POST /api/category/ – utworzenie nowej kategorii (autoryzacja wymagana),
 - DELETE /api/category/<category_id> – usunięcie kategorii (autoryzacja wymagana).
- **Relacje wydarzeń i kategorii (/api/event-category)**
 - GET /api/event-category/ – lista relacji wydarzeń z kategoriami,
 - POST /api/event-category/ – przypisanie wydarzenia do kategorii (autoryzacja wymagana),
 - DELETE /api/event-category/<relation_id> – usunięcie relacji (autoryzacja wymagana).
- **Lokalizacje (/api/location)**
 - GET /api/location/ – lista wszystkich lokalizacji,
 - GET /api/location/user/<user_id> – lokalizacje utworzone przez danego użytkownika,
 - GET /api/location/<uid> – szczegóły lokalizacji,
 - POST /api/location/ – utworzenie lokalizacji (autoryzacja wymagana),
 - DELETE /api/location/<uid> – usunięcie lokalizacji (autoryzacja wymagana).
- **Powiadomienia (/api/notifications)**
 - GET /api/notifications/ – lista powiadomień,
 - POST /api/notifications/ – dodanie powiadomienia (autoryzacja wymagana),
 - DELETE /api/notifications/<notification_id> – usunięcie powiadomienia (autoryzacja wymagana),

- POST /api/notifications/reminder – wysłanie e-maila z przypomnieniem o wydarzeniu (autoryzacja wymagana).
- PUT /api/notifications/<notification_id>/seen – oznacza powiadomienie o identyfikatorze `notification_id` jako przeczytane. Endpoint wymaga autoryzacji. Po udanym wykonaniu aktualizuje pole `status` danego powiadomienia w bazie danych na wartość `seen` i zwraca odpowiedni komunikat potwierdzający operację.

Wszystkie dane wymieniane z API są przesyłane w formacie **JSON**.

7.1.5. Obsługa Maili

System został zintegrowany z biblioteką **Flask-Mail**, co umożliwia wysyłanie wiadomości e-mail do użytkowników aplikacji. Obsługa wiadomości została zaimplementowana w module `utils.mailer`. W zależności od akcji wykonanej przez użytkownika, aplikacja wysyła różne typy wiadomości:

- potwierdzenie rezerwacji lub rezygnacji z wydarzenia (`send_confirmation_email`),
- przypomnienie o nadchodzącym wydarzeniu wraz z datą (`send_reminder_email`).

Wiadomości są personalizowane – zawierają imię użytkownika (jeśli jest dostępne), nazwę wydarzenia oraz dynamicznie generowaną treść zależną od typu akcji. Przykład treści e-maila z przypomnieniem:

Cześć Jan,

Przypominamy o nadchodzącym wydarzeniu: Koncert Chopinowski, które odbędzie się 20 czerwca 2025.

Do zobaczenia!
Zespół EVE.nt

Konfiguracja serwera SMTP odbywa się za pomocą zmiennych środowiskowych przechowywanych w pliku `.env`:

- `MAIL_SERVER`, `MAIL_PORT`,
- `MAIL_USERNAME`, `MAIL_PASSWORD`,
- `MAIL_USE_TLS`, `MAIL_DEFAULT_SENDER`.

7.1.6. Migracje Bazy Danych

Migracje struktury bazy danych realizowane są przy użyciu biblioteki **Flask-Migrate**, która integruje SQLAlchemy z narzędziem **Alembic**. Pozwala to na wygodne zarządzanie wersjonowaniem schematu bazy danych, bez konieczności ręcznego modyfikowania struktury tabel.

W module `__init__.py`, przy uruchomieniu aplikacji wykonywane są następujące operacje:

- Inicjalizacja obiektów `db` i `migrate`,
- Wywołanie `db.create_all()` i `db.Model.metadata.create_all()` — tworzące schemat bazy przy pierwszym uruchomieniu,
- Przypisanie opcji połączenia do silnika bazy danych: rozmiar puli połączeń, czas oczekiwania, itp.

Dodatkowo przygotowano możliwość zarządzania migracjami z linii polecień (komendy `flask db init`, `flask db migrate`, `flask db upgrade`), co umożliwia łatwe śledzenie zmian w modelach danych i ich propagację do środowisk deweloperskich i produkcyjnych.

Dzięki zastosowaniu tego podejścia, baza danych pozostaje spójna z kodem aplikacji, a jej modyfikacje mogą być wprowadzane w sposób bezpieczny i kontrolowany.

7.1.7. Testowanie

W celu zapewnienia poprawności działania aplikacji backendowej, zaimplementowano testy jednostkowe obejmujące kluczowe moduły systemu. Testy zostały przygotowane z wykorzystaniem biblioteki `pytest` i obejmują zarówno testy logiki biznesowej, jak i poprawności odpowiedzi endpointów REST API.

Testy znajdują się w katalogu `tests/`, są to testy jednostkowe logiki z poziomu warstwy serwisowej i pomocniczej (np. generowanie tokenów, walidacja wejścia). W celu testowania endpointów wykorzystano klienta

testowego Flask (`app.test_client()`), który umożliwia wykonywanie żądań HTTP do aplikacji bez konieczności jej uruchamiania na serwerze.

Każdy test wykonuje:

- inicjalizację tymczasowej instancji bazy danych lub aplikacji Flask;
- wykonanie operacji (rejestracja użytkownika, tworzenie wydarzenia);
- weryfikację statusu odpowiedzi oraz zawartości zwracanego JSON-a;
- czyszczenie danych po wykonaniu testu (fixture `teardown`);

Przykładowy test rejestracji użytkownika:

```
def test_register_user(client):
    response = client.post("/auth/register", json={
        "username": "testuser",
        "email": "test@example.com",
        "password": "securepass123",
        "first_name": "Test",
        "last_name": "User"
    })
    assert response.status_code == 201
    assert response.json["message"] == "User created successfully"
```

Testy uruchamiane są lokalnie za pomocą komendy:

```
pytest tests/
```

Zaimplementowane testy stanowią podstawę do dalszego rozwoju testów regresyjnych.

7.1.8. Konteneryzacja

Aplikacja backendowa została w pełni przygotowana do uruchamiania w środowisku kontenerowym Docker, co ułatwia jej wdrożenie, testowanie i przenoszenie między środowiskami.

Główny plik `Dockerfile` bazuje na lekkim obrazie `python:3.12-slim` i definiuje wszystkie kroki niezbędne do zbudowania kontenera:

- instalacja zależności systemowych wymaganych przez niektóre biblioteki (m.in. `gcc`, `libpq-dev`),
- skopiowanie i instalacja zależności Pythona z pliku `requirements.txt`,
- ustawienie katalogu roboczego `/project` oraz zmiennej `PYTHONPATH`,
- skopiowanie kodu aplikacji (folder `app/` oraz plik `run.py`),
- wystawienie portu 8800 (na którym uruchamiana jest aplikacja),
- domyślne uruchamianie aplikacji przy pomocy polecenia `flask run`.

Konfiguracja środowiska (takie jak klucze JWT, URI bazy danych PostgreSQL, parametry debugowania) znajduje się w pliku `app/.env`, który należy dołączyć podczas uruchamiania kontenera za pomocą opcji `--env-file`.

Przykładowa zawartość pliku `.env`:

```
SECRET_KEY=secret-key
REFRESH_SECRET_KEY=refresh-secret-key
REFRESH_TOKEN_AGE=1440
ACCESS_TOKEN_AGE=30
SQLALCHEMY_DATABASE_URI=postgresql://user:pass@host.docker.internal:5432/user
DEBUG=True
```

Dla pełnego uruchomienia środowiska można wykorzystać lokalną instancję PostgreSQL (np. przez osobny kontener):

```
docker run -itd \
    -e POSTGRES_USER=user \
    -e POSTGRES_PASSWORD=pass \
    -p 5432:5432 \
```

```
-v pgdata:/var/lib/postgresql/data \
--name db postgres
```

Po zbudowaniu obrazu:

```
docker build -t uaim_backend .
```

Można uruchomić aplikację backendową:

```
docker run --rm -p 8800:8800 \
--env-file app/.env \
--name uaim_backend \
uaim_backend
```

Aplikacja będzie dostępna pod adresem: <http://localhost:8800>.

W wersji produkcyjnej istnieje możliwość zamiany `flask run` na `gunicorn`, co zapewnia lepszą wydajność i obsługę wielu wątków:

```
CMD ["gunicorn", "-b", "0.0.0.0:8800", "app:create_app()"]
```

Dzięki zastosowanej konteneryzacji, uruchomienie backendu sprowadza się do kilku komend, co znaczaco ułatwia jego wdrożenie zarówno lokalnie, jak i na serwerze produkcyjnym.

8. Opis Plików Konfiguracyjnych Docker

W celu konteneryzacji aplikacji wykorzystaliśmy dedykowane pliki Dockerfile dla części backendowej oraz frontendowej. Umożliwiają one stworzenie spójnych i przenośnych środowisk uruchomieniowych dla poszczególnych komponentów systemu.

8.1. Dockerfile dla Aplikacji Backendowej

Plik **Dockerfile** (Wydruk 1) definiuje obraz kontenera dla naszej aplikacji backendowej, napisanej w Pythonie z wykorzystaniem frameworka Flask.

- **Obraz bazowy:** Jako podstawę wykorzystaliśmy oficjalny obraz `python:3.12-slim`, który jest lekką dystrybucją Pythona.
- **Katalog roboczy:** Ustawiony na `/project` wewnątrz kontenera.
- **Instalacja zależności systemowych:** Aktualizacja listy pakietów oraz instalacja `gcc` i `libpq-dev`, które są wymagane przez niektóre biblioteki Python.
- **Instalacja zależności Python:** Kopiowany jest plik `requirements.txt`, a następnie instalowane są wszystkie zdefiniowane w nim biblioteki Python za pomocą `pip`.
- **Kopiowanie kodu aplikacji:** Kopiowany jest katalog `app` zawierający logikę aplikacji oraz plik `run.py` służący do jej uruchomienia.
- **Ekspozycja portu:** Kontener będzie nasłuchiwał na porcie 8800.
- **Zmienna środowiskowa:** Ustawienie zmiennej `FLASK_APP` na `run.py`.
- **Polecenie uruchomieniowe:** Domyślnie, dla celów deweloperskich, aplikacja jest uruchamiana za pomocą wbudowanego serwera Flask (`python3 -m flask run`). W pliku znajduje się również zakomentowana alternatywna komenda dla środowiska produkcyjnego, wykorzystująca serwer `gunicorn`.

```
1 FROM python:3.12-slim
2
3 WORKDIR /project
4
5 RUN apt-get update && apt-get install -y gcc libpq-dev && rm -rf /var/lib/apt/lists/*
6
7 COPY requirements.txt /project/
8 RUN pip install --no-cache-dir -r requirements.txt
9
10 COPY app /project/app
11 COPY run.py /project/
12
13 EXPOSE 8800
14
15 ENV FLASK_APP=run.py
16
17 # For development:
18 CMD ["python3", "-m", "flask", "run", "--host=0.0.0.0", "--port=8800"]
19
20 # For production, uncomment and use gunicorn:
21 # CMD ["gunicorn", "-b", "0.0.0.0:8800", "app:create_app()"]
```

Listing 1: Zawartość pliku **Dockerfile** dla aplikacji backendowej.

8.2. Dockerfile dla Aplikacji Frontendowej (Node.js/React)

Plik **Dockerfile** (Wydruk 2) służy do budowy obrazu kontenera dla aplikacji frontendowej, opartej na Node.js i React z wykorzystaniem Vite jako narzędzia budującego.

- **Obraz bazowy:** Wykorzystano oficjalny, lekki obraz `node:20-alpine`, zawierający środowisko Node.js.
- **Katalog roboczy:** Ustawiony na `/app` wewnątrz kontenera.
- **Kopiowanie plików konfiguracyjnych i instalacja zależności:** Najpierw kopiowane są pliki `package.json` oraz `package-lock.json`, a następnie uruchamiana jest komenda `npm install` w celu pobrania wszystkich zależności projektu.
- **Kopiowanie kodu aplikacji:** Pozostała część kodu źródłowego aplikacji jest kopowana do katalogu roboczego.
- **Ekspozycja portu:** Domyślny port serwera deweloperskiego Vite, czyli `5173`, jest eksponowany na zewnątrz kontenera.
- **Polecenie uruchomieniowe:** Aplikacja jest uruchamiana w trybie deweloperskim za pomocą polecenia `npm run dev` z dodatkowymi flagami `--host 0.0.0.0`, aby serwer był dostępny z zewnątrz kontenera.

```
1 # Use official Node.js LTS image
2 FROM node:20-alpine
3
4 # Set working directory
5 WORKDIR /app
6
7 # Copy package.json and package-lock.json
8 COPY package*.json .
9
10 # Install dependencies
11 RUN npm install
12
13 # Copy the rest of the app source code
14 COPY . .
15
16 # Expose Vite default port
17 EXPOSE 5173
18
19 # Start the development server
20 CMD ["npm", "run", "dev", "--", "--host", "0.0.0.0"]
```

Listing 2: Zawartość pliku `Dockerfile.txt` dla aplikacji frontendowej.

9. Podsumowanie

Projekt portalu rezerwacji wydarzeń EVE.NT został zrealizowany zgodnie z założeniami, obejmując implementację aplikacji webowej, aplikacji mobilnej oraz backendu. Całość została przygotowana do wdrożenia w środowisku konteneryzowanym Docker, co ułatwia przenośność i zarządzanie systemem.

9.1. Możliwość Wdrożenia Komercyjnego na Serwerze VPS

Aplikacja została zaprojektowana i zaimplementowana z myślą o możliwości uruchomienia jej w środowisku produkcyjnym, zbliżonym do komercyjnego. W ramach testów i demonstracji, udało nam się z powodzeniem wdrożyć cały system na serwerze VPS, działającym pod kontrolą systemu operacyjnego Debian.

9.1.1. Uruchomienie Kontenerów na Serwerze

Na serwerze VPS, po sklonowaniu repozytorium i skonfigurowaniu zmiennych środowiskowych, aplikacja jest uruchamiana jako zestaw kontenerów Docker zarządzanych przez docker-compose. Obejmuje to kontenery dla:

- Aplikacji backendowej.
- Aplikacji frontendowej.
- Bazy danych PostgreSQL.

Poprawne działanie wszystkich kontenerów na serwerze zostało potwierdzone, co ilustruje wynik polecenia docker ps (Wydruk 3).

CONTAINER ID	IMAGE	STATUS	PORTS	NAMES
34678a414515	uaim_website	Up 2 hours	0.0.0.0:5173->5173/tcp, :::5173->5173/tcp	uaim_website
aaa223ee7ae4	uaim_backend	Up 2 hours	0.0.0.0:8800->8800/tcp, :::8800->8800/tcp	uaim_backend
b8e597a81116	postgres	Up 2 hours	0.0.0.0:5432->5432/tcp, :::5432->5432/tcp	db

Listing 3: Wynik polecenia docker ps pokazujący działające kontenery na serwerze Debian.

9.1.2. Udostępnienie w Internecie

Aby udostępnić aplikację publicznie, wykorzystaliśmy tunelowanie za pomocą usługi Cloudflare. Pozwoliło to na bezpieczne wystawienie naszych lokalnie uruchomionych na VPS kontenerów Docker do Internetu pod dedykowanymi adresami:

- Aplikacja webowa jest dostępna pod adresem: <https://uaim.karoada.ovh>
- API backendowe jest dostępne pod adresem: <https://uaim-api.karoada.ovh>

Konfiguracja tuneli Cloudflare została przedstawiona na Rys. 29.

uaim-api.karoada.ovh	*	http://192.168.1.184:8800
uaim.karoada.ovh	*	http://192.168.1.184:5173

Rys. 29: Konfiguracja publicznych nazw hostów w panelu Cloudflare wskazująca na odpowiednie porty usług na serwerze VPS.

9.1.3. Automatyzacja Wdrożenia na Serwerze

W celu usprawnienia procesu aktualizacji aplikacji na serwerze VPS, przygotowaliśmy skrypt wdrożeniowy `deploy_uaim.sh`. Skrypt ten automatyzuje następujące kroki:

1. Przechodzi do katalogu projektu na serwerze.
2. Pobiera najnowszą wersję kodu aplikacji z głównej gałęzi (`main`) repozytorium Git na GitHubie, resetując lokalne zmiany.
3. Podmienia pliki konfiguracyjne na wersje produkcyjne:
 - Plik konfiguracyjny Vite (`vite.config.js`) dla aplikacji frontendowej.
 - Plik zmiennych środowiskowych (`.env`) dla aplikacji backendowej.
4. Uruchamia główny skrypt `deploy.sh` (znajdujący się w repozytorium projektu), który jest odpowiedzialny za przebudowanie i ponowne uruchomienie kontenerów Docker za pomocą `docker`.

Dzięki temu proces aktualizacji aplikacji na serwerze jest szybki, powtarzalny i mniej podatny na błędy manualne.

```
----- deploy_uaim.sh -----  
#!/bin/bash  
set -e # Przerwij wykonywanie skryptu w przypadku błędu  
  
# --- Konfiguracja ---  
# Nazwa katalogu projektu  
PROJECT_DIR_NAME="uaim_project"  
# Gałąź do wdrożenia  
DEPLOY_BRANCH="main"  
  
# Ścieżka do katalogu, w którym znajduje się ten skrypt  
SCRIPT_DIR="$(cd "${(dirname "${BASH_SOURCE[0]}")}" &>/dev/null && pwd)"  
  
# Pełna ścieżka do katalogu projektu  
PROJECT_PATH="${SCRIPT_DIR}/${PROJECT_DIR_NAME}"  
  
# Ścieżki do plików źródłowych (w katalogu ze skryptem deploy_uaim.sh)  
SOURCE_VITE_CONFIG_FILE="${SCRIPT_DIR}/vite.config.prod.js"  
SOURCE_BACKEND_ENV_FILE="${SCRIPT_DIR}/backend.env.prod"  
  
# Ścieżki docelowe wewnątrz projektu uaim_project  
TARGET_VITE_CONFIG_PATH="${PROJECT_PATH}/website/vite.config.js"  
TARGET_BACKEND_ENV_PATH="${PROJECT_PATH}/backend/app/.env"  
  
echo "---- Rozpoczynanie procesu wdrożenia dla projektu ${PROJECT_DIR_NAME} ----"  
  
# --- Sprawdzenie warunków wstępnych ---  
if [ ! -d "${PROJECT_PATH}" ]; then  
    echo "BŁĄD: Katalog projektu '${PROJECT_PATH}' nie został znaleziony."  
    echo "Upewnij się, że ten skrypt znajduje się w katalogu nadziednym '${PROJECT_DIR_NAME}'. "  
    exit 1  
fi  
  
if [ ! -f "${SOURCE_VITE_CONFIG_FILE}" ]; then  
    echo "BŁĄD: Produkcyjny plik Vite '${SOURCE_VITE_CONFIG_FILE}' nie został znaleziony."  
    echo "Proszę utwórz go w tym samym katalogu co ten skrypt."  
    exit 1  
fi  
  
if [ ! -f "${SOURCE_BACKEND_ENV_FILE}" ]; then  
    echo "BŁĄD: Produkcyjny plik .env dla backendu '${SOURCE_BACKEND_ENV_FILE}' nie został  
        → znaleziony."  
    echo "Proszę utwórz go w tym samym katalogu co ten skrypt."  
    exit 1
```

```

fi

# 1. Przejdź do katalogu projektu
echo ""
echo "Przechodzenie do katalogu projektu: ${PROJECT_PATH}"
cd "${PROJECT_PATH}" || { echo "BŁĄD: Nie można przejść do katalogu projektu ${PROJECT_PATH}"; exit 1;
}

# 2. Operacje Git
echo ""
echo "Pobieranie najnowszych zmian z repozytorium (origin)..."
git fetch origin

echo "Resetowanie lokalnych zmian do stanu z origin/${DEPLOY_BRANCH}..."
git reset --hard "origin/${DEPLOY_BRANCH}"
# git pull origin "${DEPLOY_BRANCH}" # Opcjonalnie, reset --hard powinien wystarczyć

# 3. Kopiowanie produkcyjnego pliku vite.config.js
echo ""
echo "Kopiowanie produkcyjnej konfiguracji Vite..."
cp "${SOURCE_VITE_CONFIG_FILE}" "${TARGET_VITE_CONFIG_PATH}"
echo "Skopiowano '${SOURCE_VITE_CONFIG_FILE}' do '${TARGET_VITE_CONFIG_PATH}'"

# 4. Kopiowanie produkcyjnego pliku .env dla backendu
echo ""
echo "Kopiowanie produkcyjnego pliku .env dla backendu..."
# Upewnij się, że katalog docelowy istnieje
mkdir -p "$(dirname "${TARGET_BACKEND_ENV_PATH}")"
cp "${SOURCE_BACKEND_ENV_FILE}" "${TARGET_BACKEND_ENV_PATH}"
echo "Skopiowano '${SOURCE_BACKEND_ENV_FILE}' do '${TARGET_BACKEND_ENV_PATH}'"

# 5. Uruchomienie istniejącego skryptu deploy.sh
echo ""
echo "Nadawanie uprawnień wykonywalnych dla skryptu deploy.sh (jeśli potrzebne)..."
chmod +x deploy.sh

echo "Uruchamianie głównego skryptu wdrożeniowego (deploy.sh)..."
./deploy.sh

echo ""
echo "--- Proces wdrożenia zakończony pomyślnie! ---"

```

Uważamy, że zrealizowany projekt, wraz z przedstawionym procesem wdrożenia i konteneryzacją na maszynie wirtualnej z systemem Debian, stanowi solidną podstawę do dalszego rozwoju i potencjalnego uruchomienia w pełnoprawnym środowisku komercyjnym.