Analiza systemu powiadomień on-line o eventach realizowanych w mieście

Projekt zespołowy

Filip Kamiński, Kamil Wielgosz, Remigiusz Frankiewicz

Z712

Eventura.pl

Dokumentacja - https://github.com/ABSF/eventura/tree/develop/docs

Frontend - https://github.com/ABSF/eventura

Backend - https://github.com/ABSF/eventura-server

1. Przedstawienie koncepcji systemu

Na rynku funkcjonuje wiele portali, które zazwyczaj ograniczają się do filtrowania i informowania użytkowników o specyficznym typie wydarzeń (np. Tylko koncerty, tylko wydarzenia teatralne etc.)
Założeniem projektu jest utworzenie systemu/aplikacji internetowej informującej użytkowników o wydarzeniach społecznych i kulturalnych w ich okolicy.

Nowo tworzony system ma być z założenia agregatorem wydarzeń ze wszystkich możliwych kategorii. Treści, które będą przedstawiane na stronie będą tworzone za pomocą dwóch metod:

- Wpisy generowane od podstaw
- Wpisy pobierane bezpośrednio od organizatorów wydarzeń za pomocą API i umieszczane bezpośrednio w systemie

Serwis eventura.pl umożliwiać będzie wyszukiwanie i filtrowanie wydarzeń w sposób prosty i przyjazny użytkownikowi na podstawie:

- Lokalizacji
- Kategorii
- Daty
- Pola wyszukiwania (słowa kluczowe)

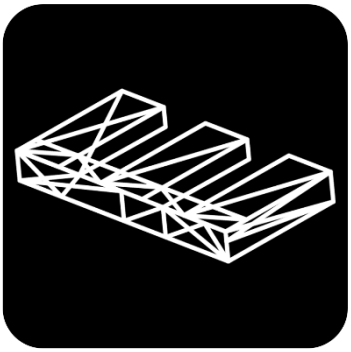
Zaimplementowane mechanizmy będą miały na celu skrócić czas wyszukiwania co skutkuje szybszym dostępem do informacji i jednocześnie przekłada się na pozytywny odbiór strony i dotarcie do jak największej liczby użytkowników (bez względu na poziom zaawansowania w obyciu z systemami internetowymi).

Serwis dodatkowo umożliwiać będzie subskrypcję interesujących użytkowników nowych wydarzeń za pomocą newsletteru oraz powiadomień push w przeglądarce internetowej - które w ostatnim czasie są bardzo popularnym sposobem na zapewnienie sobie większej liczby odwiedzin strony niż w przypadkach ,,klasycznego” marketingu internetowego.

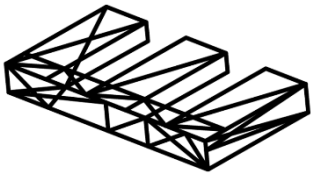
Podczas projektowania zostanie przemyślana kwestia ewentualnego przyszłego rozwoju serwisu.

Serwis pod kątem designu będzie utrzymany w motywie minimalistycznym. Formy, interfejsy oraz grafiki zawarte na stronie zostaną dobrane pod kątem uniwersalności i czytelności dla np. Różnych grup wiekowych.

Wstępna identyfikacja wizualna serwisu



Sygnet wersja kolorystyczna nr 1



Sygnet wersja kolorystyczna nr 2

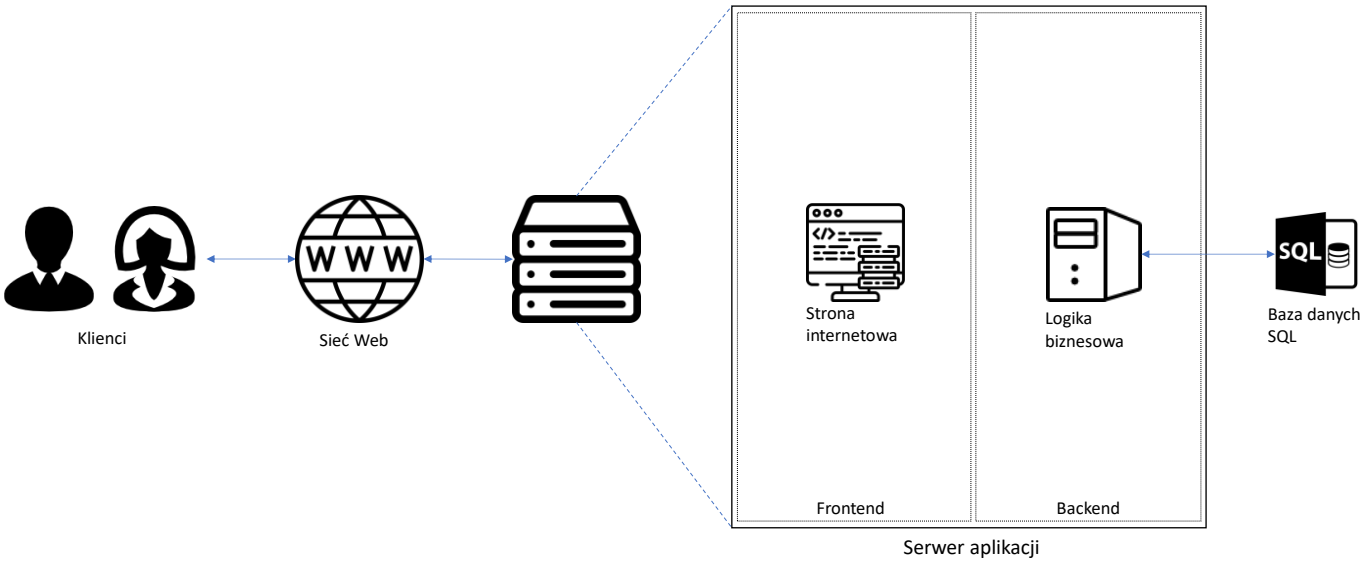
Wariacja kolorystyczna sygnetu

Typografia wersja kolorystyczna nr 1

Typografia wersja kolorystyczna nr 2

Wariacja kolorystyczna typografii

2. Model architektury systemu



Powyżej został przedstawiony diagram architektury systemu.

Klient z wykorzystaniem sieci web ma nieograniczona możliwość łączenia się oraz korzystania z systemu internetowego. Serwer aplikacji złożony jest z dwóch głównych części: frontend i backend. Rozdzielenie warstwy prezentacji od logiki biznesowej oraz magazynu danych pomaga zachować wyraźny podział odpowiedzialności każdej z tych warstw. Sprzyja to także bezpieczeństwu systemu, ponieważ walidację możemy przeprowadzać na kilku etapach – wprowadzania danych, przetwarzaniu danych oraz zapisie danych. Tym samym zwiększamy bezpieczeństwo zachowania integralności danych. System zapisuje wszystkie niezbędne informacje na temat wydarzeń w bazie danych SQL.

3. Specyfikacja funkcjonalna

- Główna idea aplikacji:

Strona/aplikacja internetowa jako agregator informacji o imprezach i wydarzeniach kulturalnych w mieście.

- Cel biznesowy:
  Dotarcie do jak największej liczby osób z wydarzeniami kulturalnymi w celu poszerzeniu wiedzy na temat różnego rodzaju rozrywki. Drugim celem jest ułatwienie użytkownikom odnalezienie dopasowanych do ich potrzeb i zainteresowań eventów zarówno masowych jak i kameralnych.

- Grupa docelowa aplikacji:

Osoby w każdym przedziale wiekowym, aktywne kulturowo

- Czynności oraz akcje w ramach aplikacji:

-wyszukiwanie po stringu/słowach kluczowych (np. nazwisko, nazwa itp..)
-wyszukiwanie po regionie (np. po miastach, województwach)
-wyszukiwanie po kategorii (koncerty, spektakle itp..)
-wyszukiwanie po dacie (zakres daty od-do)
-wpisy w formie listy lub kafelków z eventami na głównej stronie i podstronie po wyszukaniu
-newsletter na podstawie subskrypcji np. Regionu/miasta
-powiadomienia push w przeglądarce internetowej

- Ograniczenia we wstępnej fazie projektowania i implementacji

-brak możliwości logowania
-brak możliwości zakupu wejściówek/biletów na wydarzenia

- Integracja z innymi systemami:

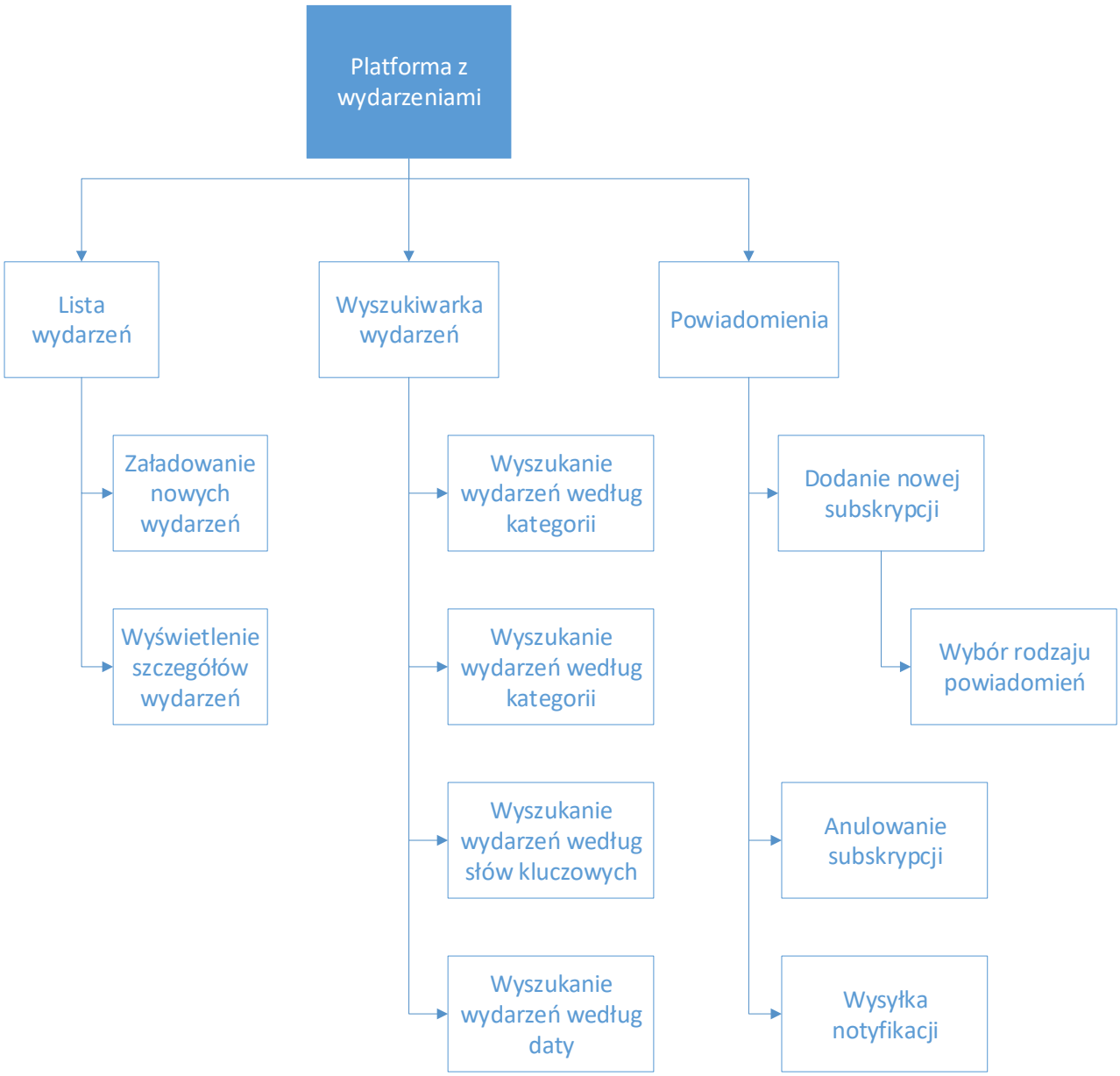
-integracja z bazą danych
-API zewnętrznych dostawców za pomocą którego istnieje możliwość szybkiego pozyskiwania informacji na temat wydarzeń

- Platformy/kompatybilność:

-responsywna wersja desktop
-responsywna wersja mobile

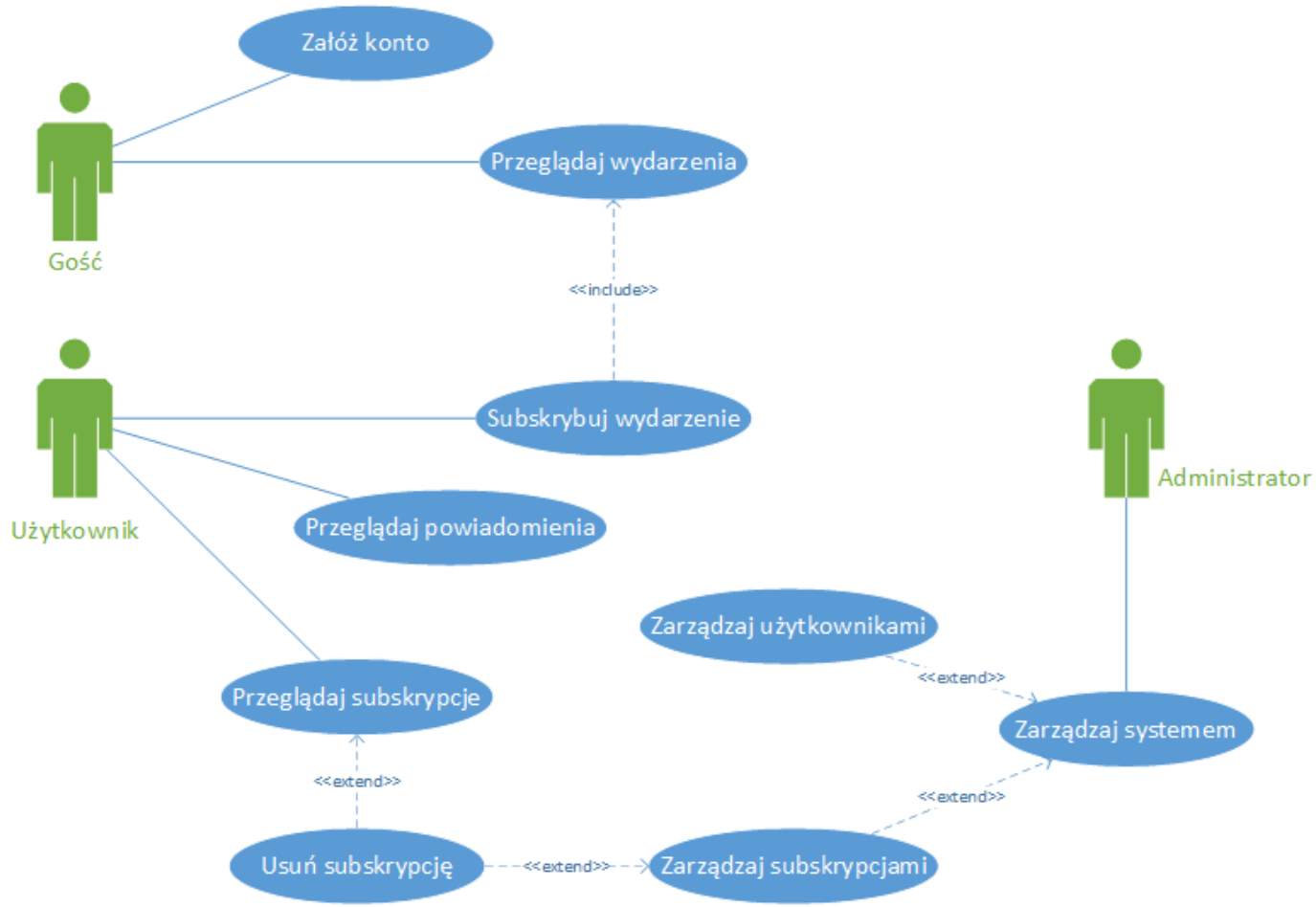
4. Diagramy UML

Diagram hierarchii funkcji:



Powyższy diagram przedstawia zarys dostępnych funkcji platformy. Będą dostępne 3 główne założenia biznesowe – prezentacja listy eventów, przeszukanie bazy eventów pod kątem określonych parametrów oraz powiadomienia o zbliżających się wydarzeniach. Na lista wydarzeń będzie zasilana ze źródła danych oraz będzie ona wyświetlana na stronie głównej. Wyszukiwarka będzie posiadała kilka zdefiniowanych filtrów. Z kolei subskrypcje użytkownik będzie mógł dodać, anulować oraz otrzymać zasubskrybowaną notyfikację.

Diagram przypadków użycia:



Aktorzy:

- Gość – klient serwisu bez zarejestrowanego konta. Ma tylko możliwość przeglądania wydarzeń oraz założenia nowego konta.
- Użytkownik – klient zalogowany do serwisu. Względem gościa może dodatkowo przeglądać aktualne subskrypcje, usuwać je jak i subskrybować nowe wydarzenia. Na podstawie aktualnych subskrypcji ma możliwość przeglądać powiadomienia.
- Administrator – użytkownik systemu zarządzający serwisem. Ma możliwość zarządzania kontami zarejestrowanych klientów oraz ich subskrypcjami.

Diagram przepływu danych:

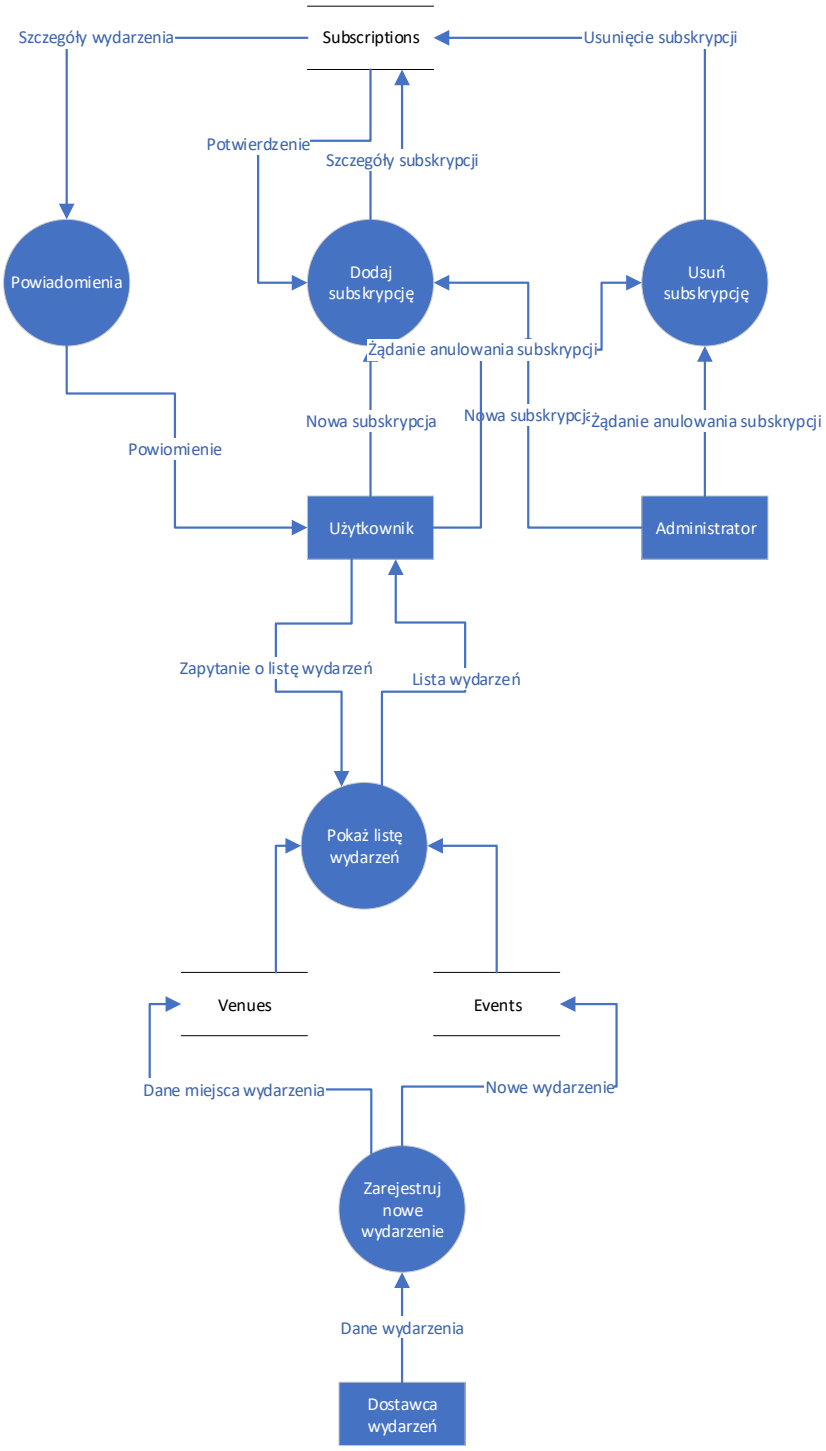
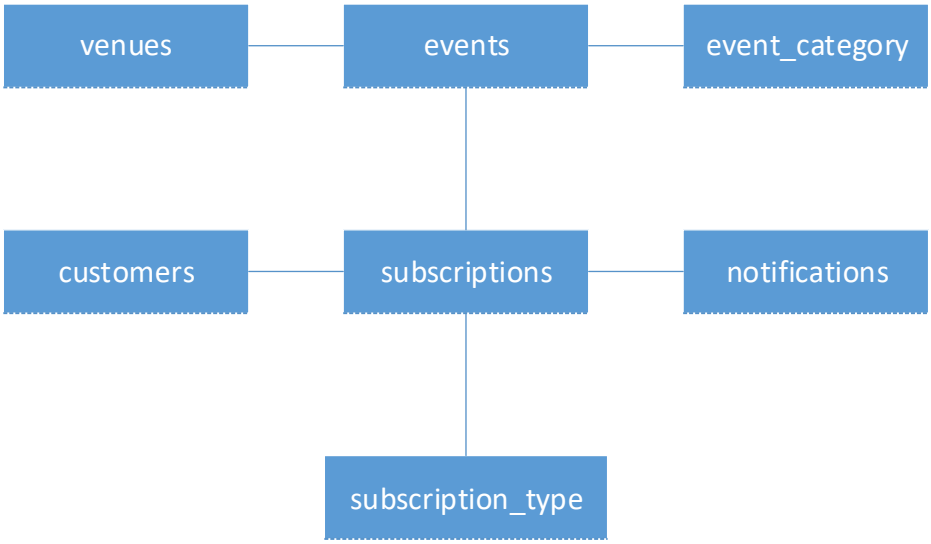


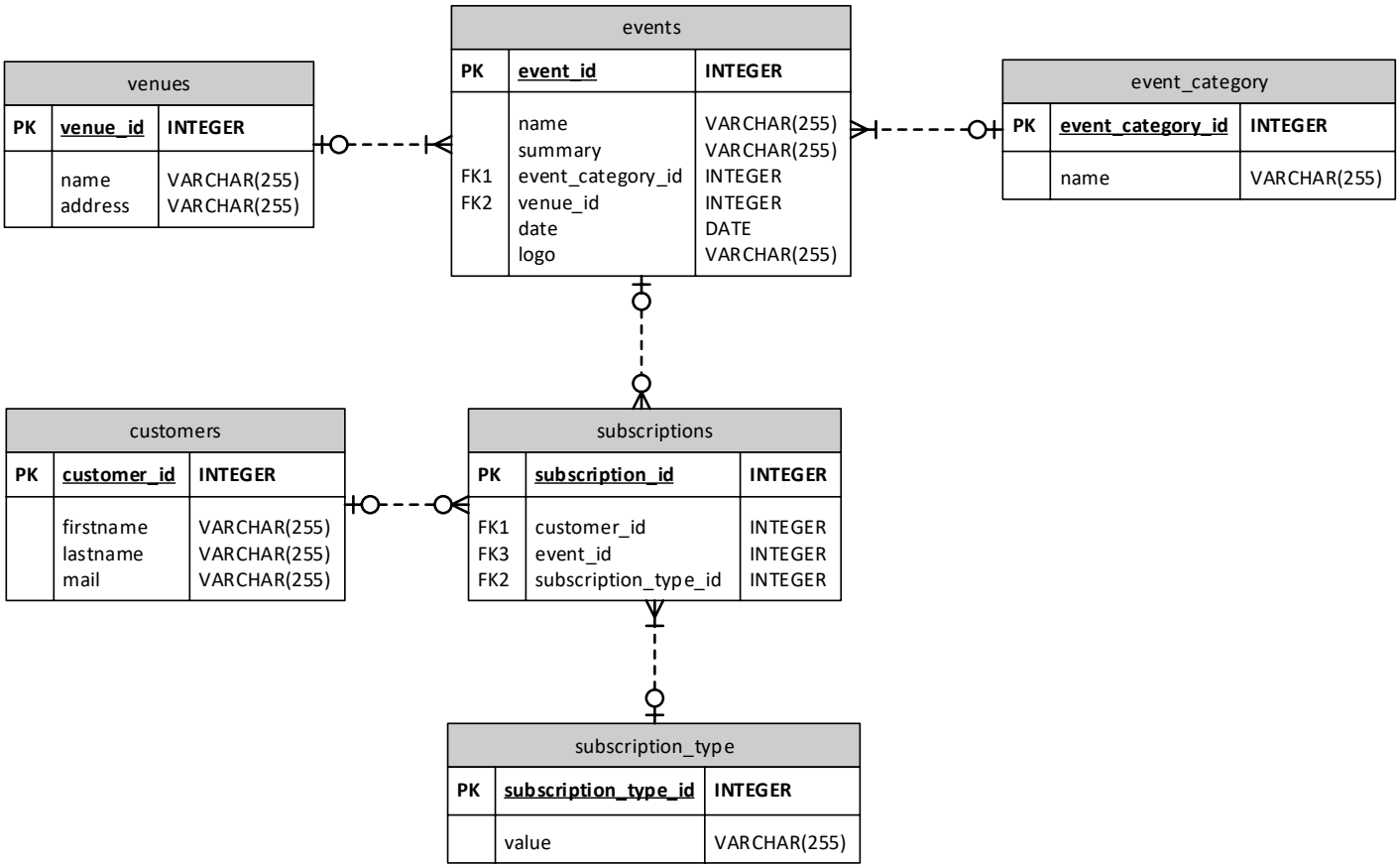
Diagram przepływu danych pokazuje ogólny zarys systemu. Funkcjonalność listy wydarzeń będzie pobierała dane z magazynów danych, które będą zasilane przez zewnętrzne źródło lub manualnie. Użytkownik może wyświetlić tę listę oraz przeszukać. Dodatkowo dla każdego eventu będzie mógł dodać, anulować oraz otrzymywać powiadomienia, które będą składowane w oddzielnym magazynie danych.

5. Model danych



Powyżej został przedstawiony podstawowy diagram związków encji, które będą wykorzystywane w systemie. Główną częścią jest encja events – zawiera ona szczegóły dotyczące każdego wydarzenia. Łączy się ona z encją słownikową event_category, w której przetrzymywane będą kategorie wydarzeń. W encji venues znajdują się miejsca w których odbywają się wydarzenia. Każdy event może zostać zasubskrybowany i każda subskrypcja jest składowana w encji subscriptions. W kolejnej encji słownikowej - subscription_type – przechowywane są rodzaje subskrypcji np. push lub mail. Informacje kto dokonał subskrypcji są przechowywane w encji customers z podstawowymi danymi użytkownika. Dodatkowo istnieje encja do celów audytowych – notifications do składowania informacji o wysłanych notyfikacjach.

6. Projekt bazy danych



Co do systemu zarządzania bazą danych wybór padł na PostgreSQL. Uzasadnieniem jest to, iż jest darmowym rozwiązaniem, co ogranicza koszty projektu. Można zdefiniować również wiele nowych, niestandardowych dla baz danych, typów danych np. JSON (np. w przyszłości można użyć go do utworzenia tabeli audytowej komunikacji z zewnętrznym API) lub UUID (np. przydatny typ danych w przypadku migracji danych – na obecnym etapie projektu nie będzie wykorzystywnay - ponieważ jest unikalny globalnie oraz ze względów bezpieczeństwa, ponieważ nie da się zgadnąć wartości UUID w porównaniu do ID zwiększanego o 1).

Tabela events będzie zawierała informację o każdym wydarzeniu dostępnym w systemie. Posiada relację wiele do jednego z tabelą venues oraz event_category. Każde wydarzenie może posiadać tylko jedną kategorię oraz może się odbywać tylko w jednym miejscu (w tym miejscu zakładamy, że wydarzenie odbywające się w dwóch terminach, w bazie danych będzie zapisane jako dwa oddzielne eventy). Atrybut event_id posiada klucz główny i służy do unikalnej identyfikacji każdego wydarzenia. Pozostałe atrybuty to:

- Name – nazwa wydarzenia
- Summary – krótki opis wydarzenia
- Event_category_id – id kateogrii wydarzenia z tabeli event_category
- Venue_id – id miejsca, w którym odbywa się wydarzenia z tabeli venues
- Date – data wydarzenia
- Logo – ścieżka lub URL do logo wydarzenia

Tabela venues zawiera informacje na temat miejsc odbywania się wydarzeń. Posiada relację jeden do wielu z tabelą events – w każdym miejscu może odbyć się wiele wydarzeń. Kluczem główny jest nałożony na atrybut venue_id.

Dodatkowo tabela będzie zawierała poniższe atrybuty:

- Name – nazwa miejsca
- Address – adres miejsca

Tabela event_category przetrzymuje kategorie wydarzeń. Atrybut event_category_id posiada klucz główny. Posiada relację 1 do wielu z tabelą events – jedna kategoria może być przypisana do wielu wydarzeń. Dodatkowo zawiera jedynie jeden atrybut:

- Name – nazwa kategorii wydarzenia

Tabela subsciptions będzie przechowywała informacje u subskrypcjach powiadomień o wydarzeniach. Posiada relację wiele do jednego z tabelą events – jedno wydarzenie może posiadać wiele subskrypcji. Z tabelą customer posiada relację wiele do jednego – jeden użytkownik może posiadać wiele subskrypcji. Kolejną relacją jest relacja wiele do jednego z tabelą subscription_type – każda subskrypcja może posiadać tylko jeden typ. Klucz główny będzie na atrybucie subscription_type_id. Oprócz tego tabela posiada jeszcze jeden atrybut:

- Customer_id – id użytkownika do którego przypisana jest subskrypcja
- Event_id – id wydarzenia, którego dotyczy subskrypcja
- Subscription_type_id – id kategorii subskrypcji

Tabela subscription_type zawiera kategorie subskrypcji. Posiada relację jeden do wielu z tabelą subscriptions – jedna kategoria może być przypisana do wielu subskrypcji. Klucz główny nałożony jest na atrybut subscription_type_id. Oprócz tego tabela posiada jeszcze jeden atrybut:

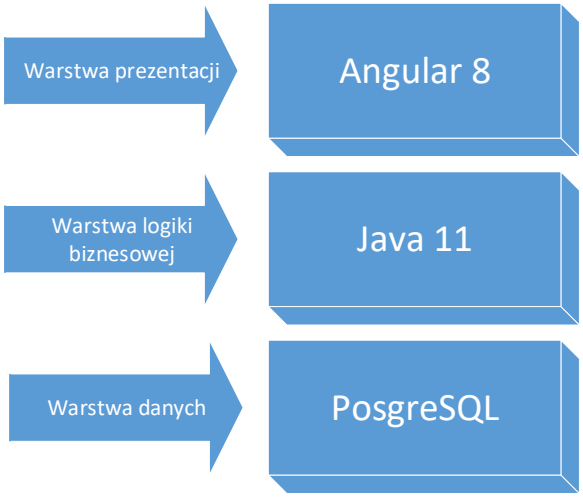
- Value – nazwa kategorii subskrypcji

Tabela customers przechowuje dane o użytkownikach systemu. Posiada relacje jeden do wielu z tabelą subscriptions – jedna osoba może posiadać wiele subskrypcji. Na atrybut customer_id będzie nałożony klucz główny. Dodatkowo będzie posiadała atrybuty:

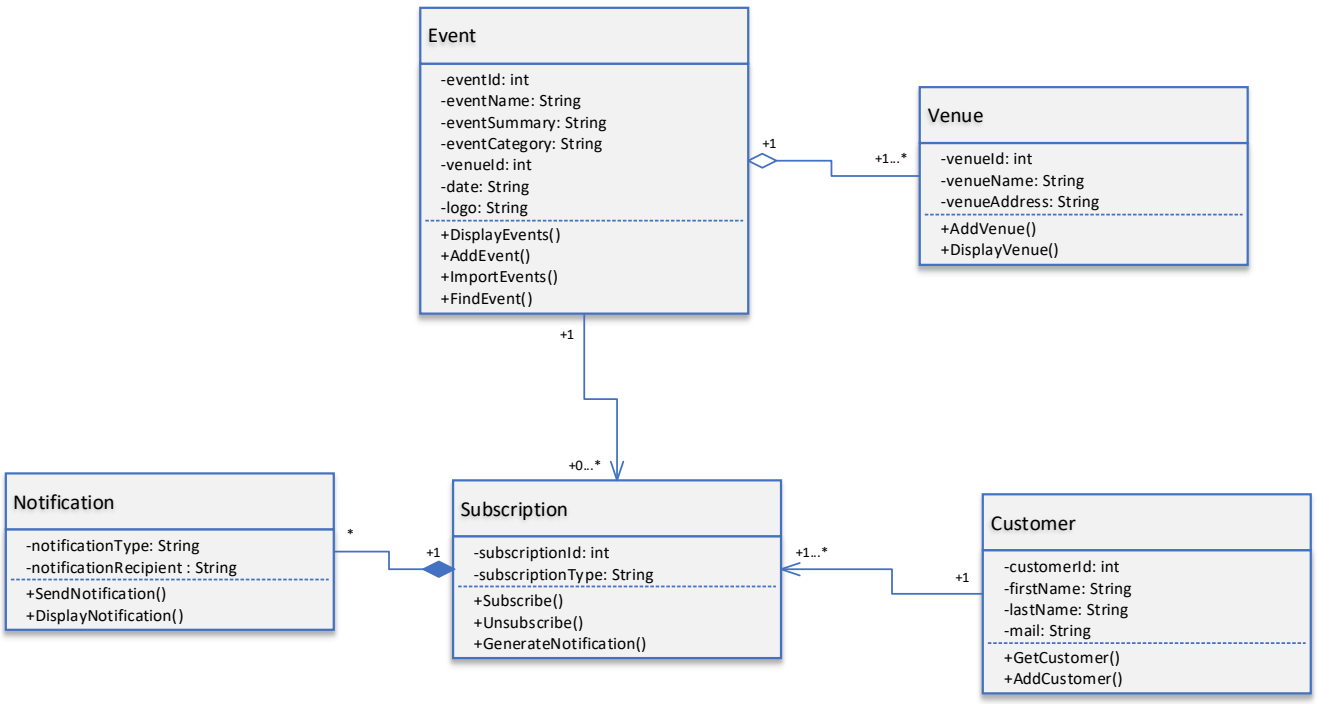
- Firstname – imię użytkownika
- Lastname – nazwisko użytkownika
- Mail – adres mailowy użytkownika

7. Projekt architektury systemu

Poniższa ilustracja przedstawia podział warstw aplikacji wraz z użytymi technologiami. Warstwa prezentacji zostanie napisana przy użyciu frameworka JavaScript – Angular. Będzie odpowiadała głównie za prezentacji treści, która otrzyma z backendu. Warstwa logiki biznesowej zostanie napisana w Javie 11 z użyciem Springa oraz hibernate. Dane do aplikacji będą przechowywane w relacyjnej bazie danych z użyciem systemu do zarządzania bazą danych – PostgreSQL.



- diagram klas

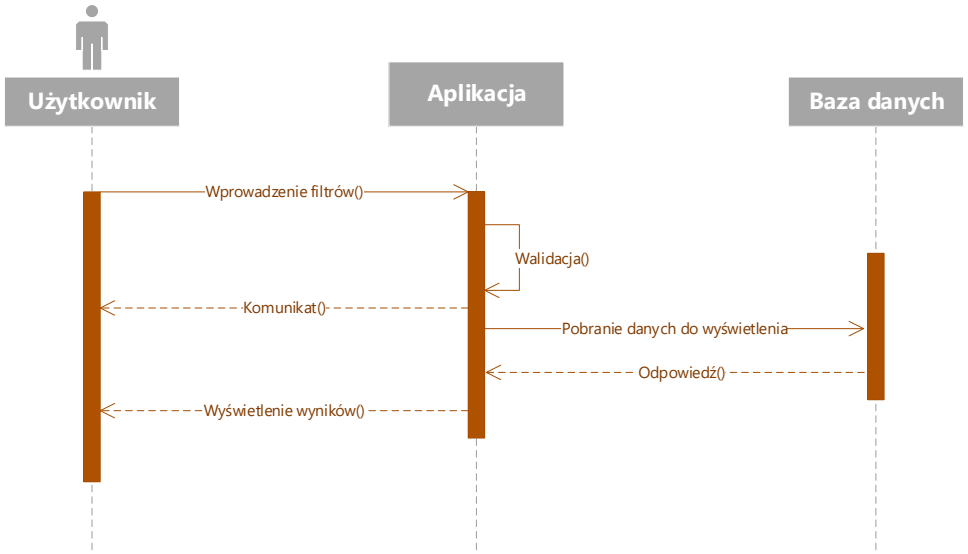


Powyższy diagram przedstawia relacje między poszczególnymi obiektami w systemie notyfikacji o lokalnych wydarzeniach. Klasa Event zawiera informacje o danym wydarzeniu. Zawiera metodę, która zwraca obiekt z informacjami potrzebnymi do wyświetlenia na froncie. Dodatkowo ta klasa zawiera metody odpowiedzialne za dodawanie i import nowych wydarzeń. W przypadku potrzeby wyszukania wydarzenia można wykorzystać metodę FindEvent. Klasa Venue potrzebna jest do tworzenia obiektów miejsc wydarzeń oraz wyświetlenie poszczególnych miejsc na liście adresów. Klasa Subscription odpowiada za dodawanie subskrypcji i generowanie notyfikacji. Klasa Notification odpowiada już za konkretną notyfikację wysyłaną do użytkownika – obiekt notyfikacji nie może istnieć bez subskrypcji. Klasa Customer to konkretni użytkownicy, którzy mogą zasubskrybować wiele wydarzeń.

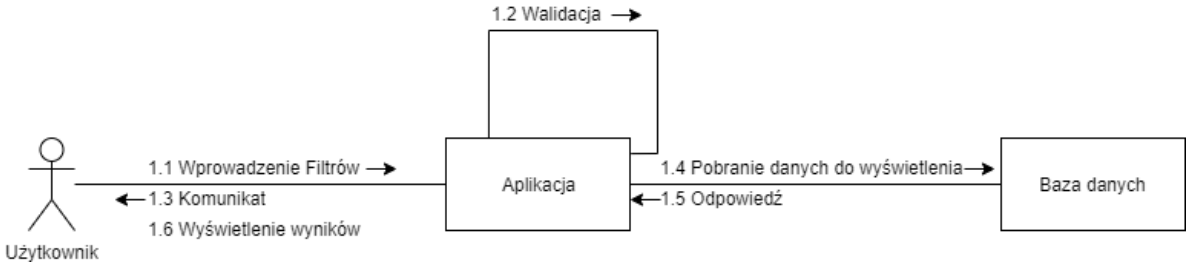
8. Projekt algorytmów

Lista wydarzeń:

W tej części systemu każdy użytkownik może zobaczyć listę wydarzeń. Na stronie głównej liczba wyników będzie ograniczona do domyślnych filtrów. W przypadku potrzeby zawężenia wyników użytkownik może podać swoje wymagania, które przejdą walidację, po czym zostaną pobrane dane do wyświetlenia, oraz nastąpi samo wyświetlenie.



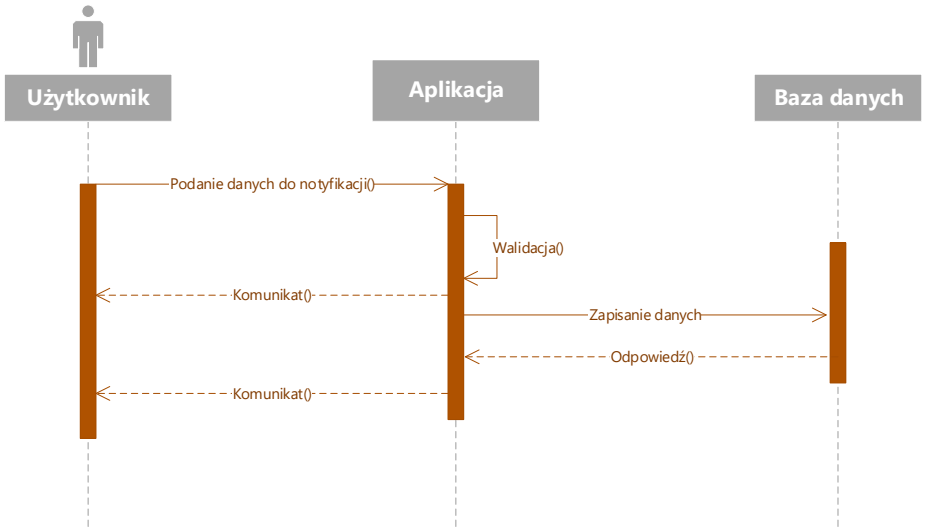
1 Diagram sekwencji - lista wydarzeń



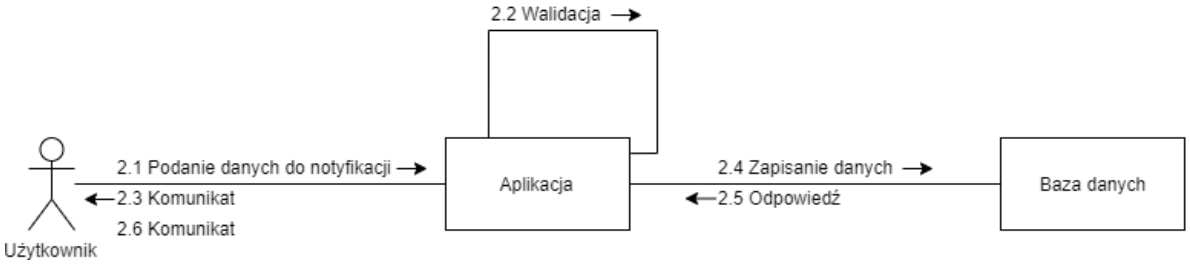
2 Diagram współpracy - lista wydarzeń

Subskrypcja powiadomień:

Do każdego nadchodzącego wydarzenia użytkownik systemu może zasubskrybować powiadomienia, aby nie zapomnieć o nim. Na początek musi podać dane do notyfikacji tj. typ powiadomienia, adres mailowy. W warstwie aplikacji następuje walidacja podanych danych i zapis do bazy danych.



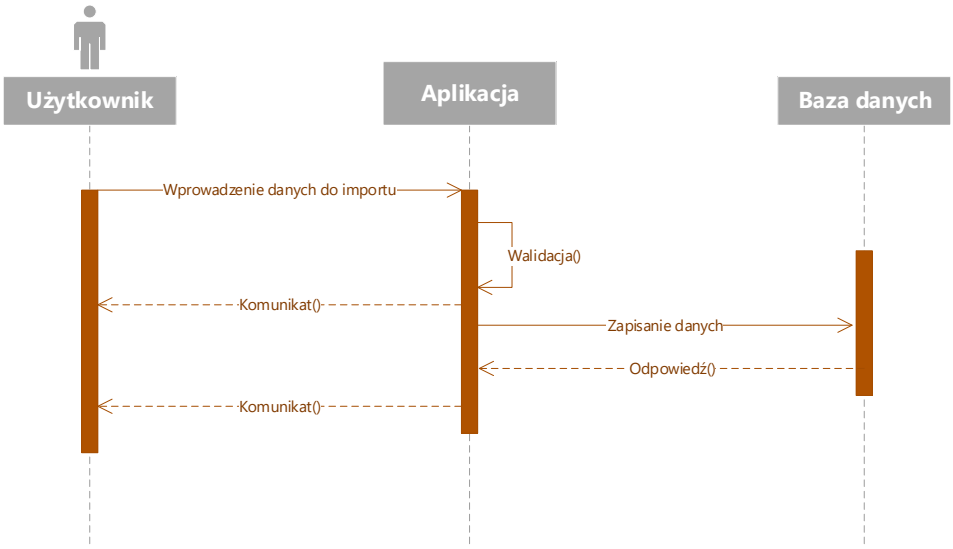
3 Diagram sekwencji - subskrypcja powiadomień



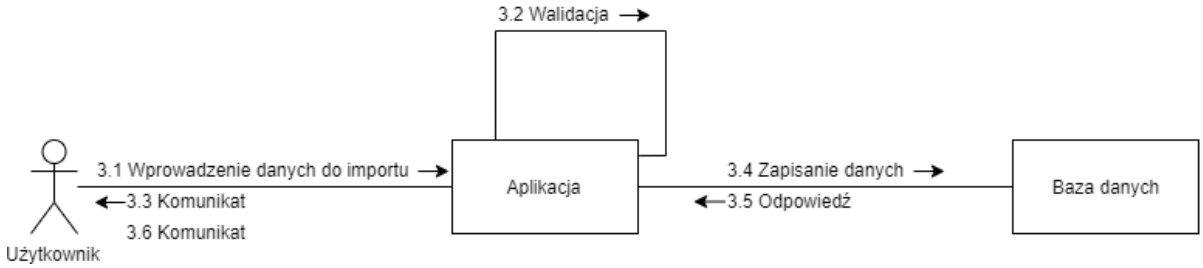
4 Diagram współpracy - subskrypcja powiadomień

Dodawanie wydarzeń:

Do systemu można dodawać wydarzenia. Może to robić administrator ręcznie lub import z zewnętrznego API. Niezależnie od źródła dane są poddawane walidacji i dopiero wtedy zapisywane do bazy, a następnie wykorzystywane do prezentacji wyników na liście wydarzeń.



5 Diagram sekwencji - dodawanie wydarzeń



6 Diagram współpracy - dodawanie wydarzeń

9. Projekt interfejsu użytkownika

Przy projektowaniu interfejsu użytkownika kluczowa jest funkcjonalność samego serwisu jak też i czytelność wszystkich informacji. W przypadku agregatorów wydarzeń liczba elementów do wyświetlenia jest duża, treści ulegają częstej zmianie (aktualizowanie obecnych wydarzeń i dodawanie nowych).

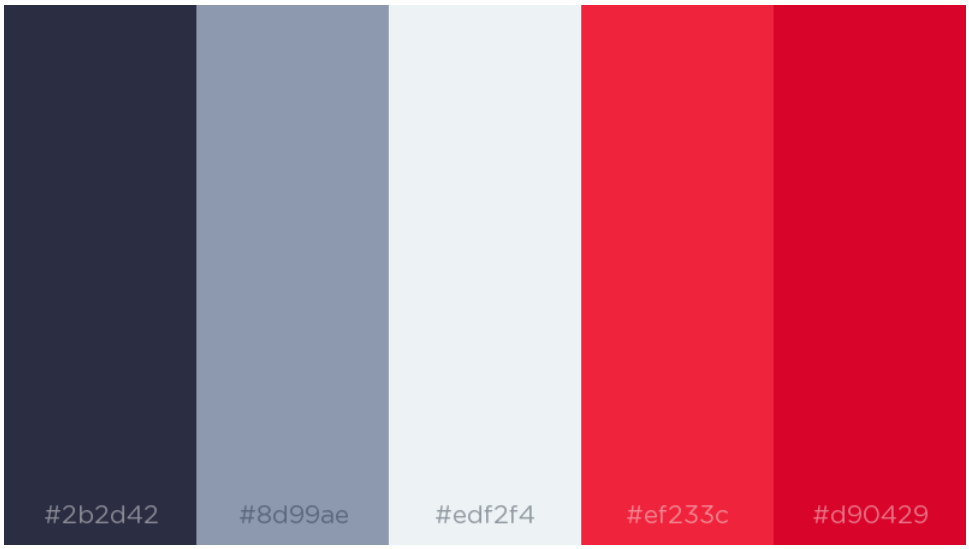
Na stronie głównej zostanie zaimplementowany kafelkowy widok elementów.

Na widokach poszczególnych wydarzeń informacje będą przedstawiane w sposób prosty i czytelny – tylko najważniejsze informacje ukażą się użytkownikom ,, na pierwszy rzut oka" w celu ograniczenia przeładowania informacją.

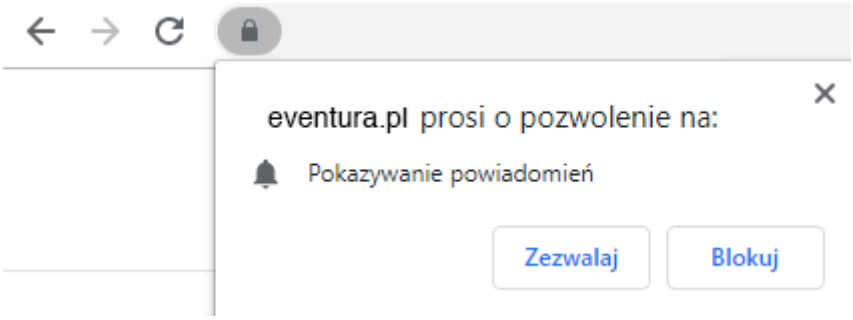
Serwis będzie umożliwiał również subskrypcję na zasadach newslettera. Zapis odbywać się będzie poprzez wypełnienie odpowiedniego formularza.

Wyszukiwanie wydarzeń w serwisie odbywać się będzie za pośrednictwem wyszukiwarki dostępnej z poziomu każdej podstrony (prawy górny róg strony) jak też i za pośrednictwem dedykowanej podstrony wyszukiwania.

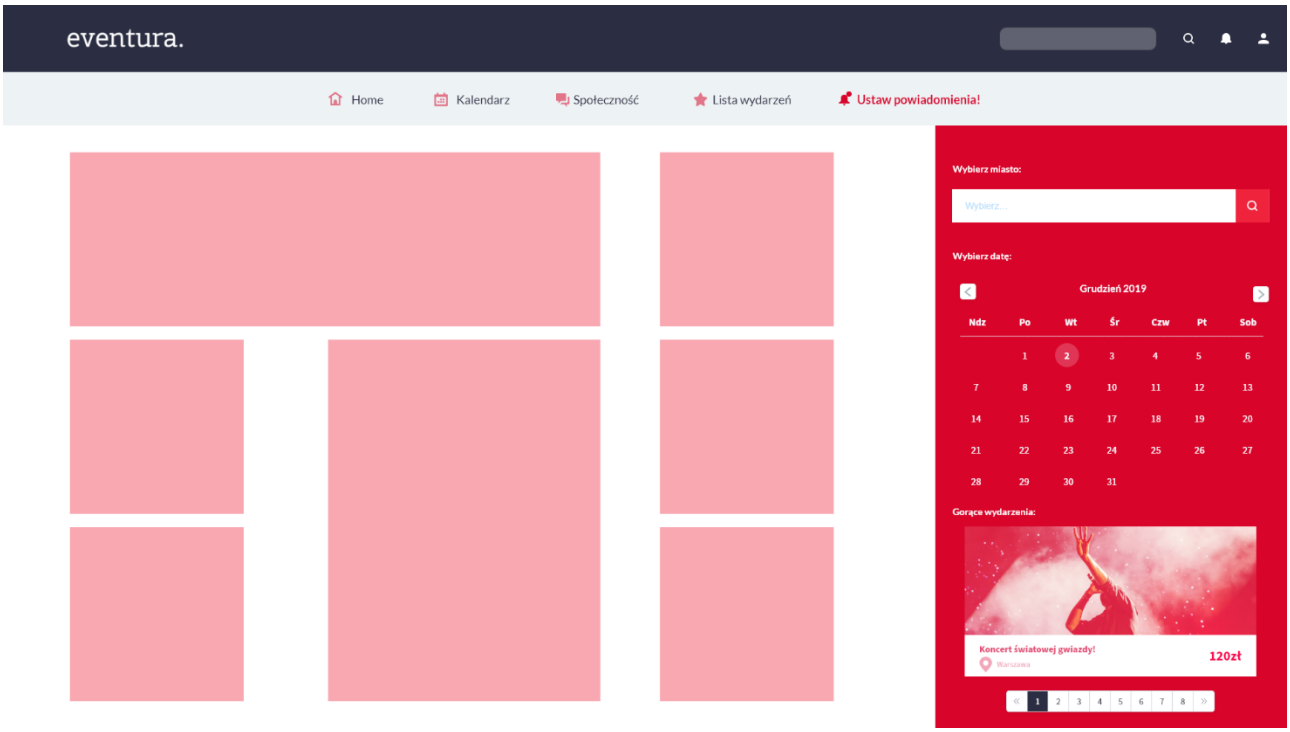
Mechanizm wyświetlania powiadomień push dla użytkownika (razem z możliwością akceptacji otrzymywania powiadomień) będzie obsługiwany przez konkretne przeglądarki internetowe.



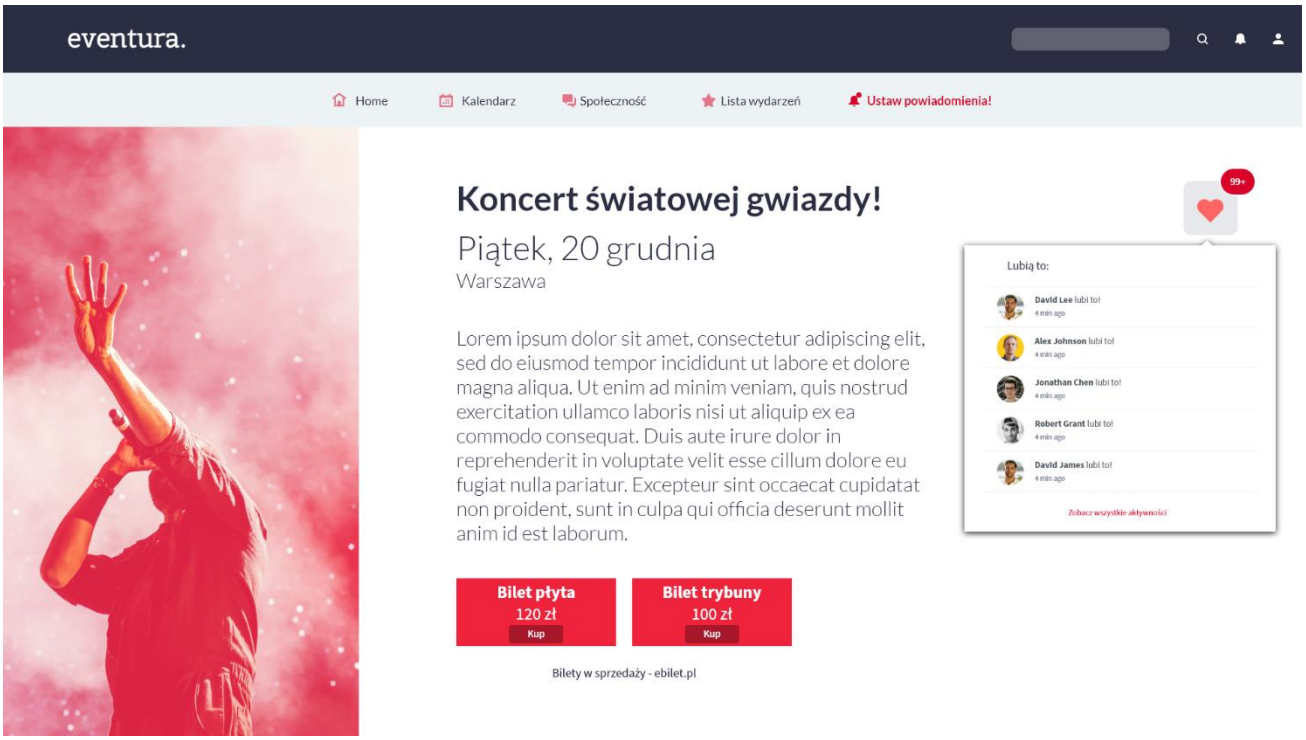
1. Paleta kolorów (HEX) użyta podczas projektowania wyglądu serwisu



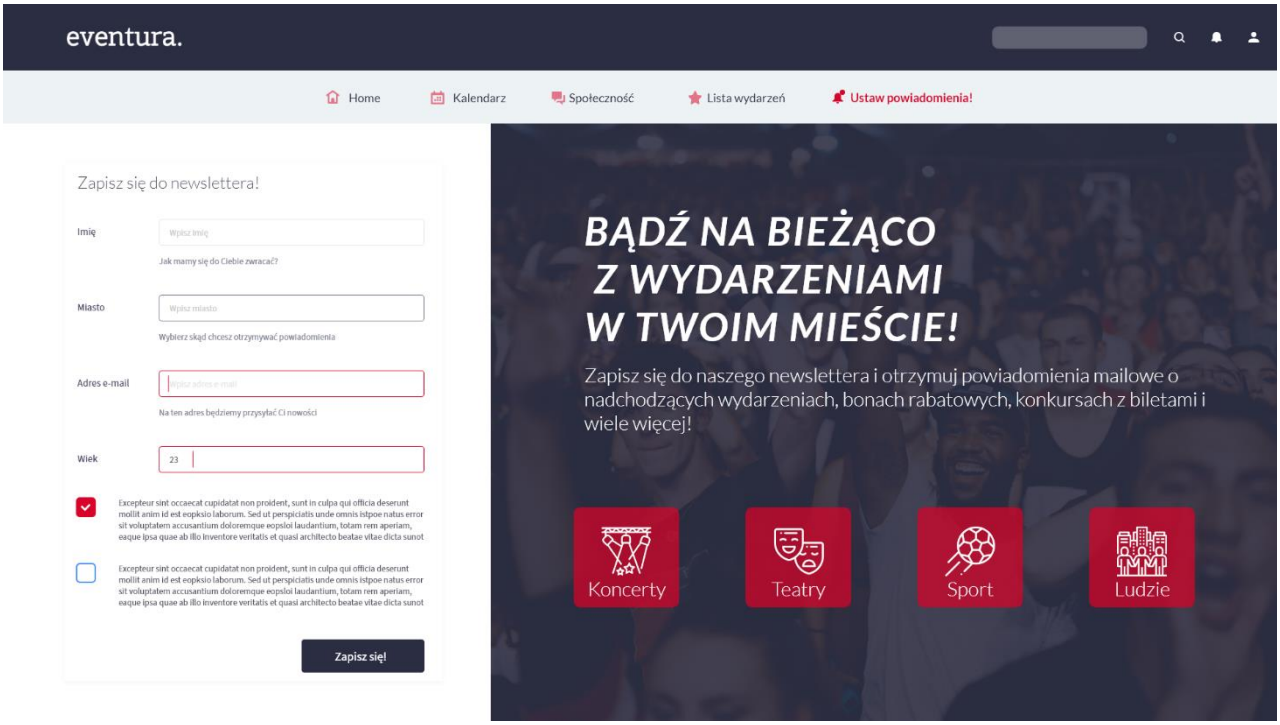
2. *Okno akceptacji otrzymywania powiadomień*



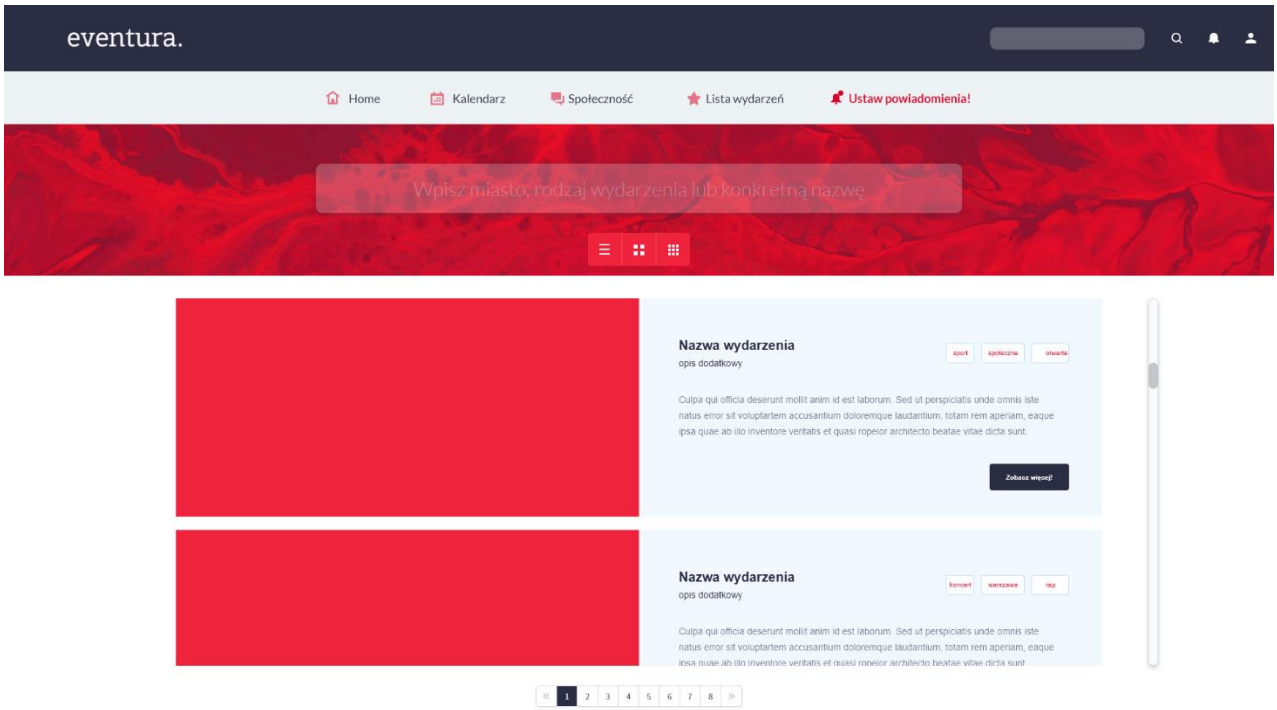
3. *Makieta strony głównej*



4. Makieta widoku wydarzenia



5. Makieta widoku subskrypcji newslettera



6. Makieta widoku wyszukiwania zaawansowanego

10. Projekt testów kontrolnych

Aby sprostać wygórowanym oczekiwaniom klienta oraz zachować wymagane cechy portalu należy dokładnie przetestować stworzoną aplikację.

Testy aplikacji dzielą się na kilka etapów:

Testy jednostkowe

Testy w ramach których testowane są poszczególne metody frontend'owe jak i back-end'owe. Przeprowadzane głównie już w etapie developmentu aplikacji w celu zmiejszenia czasu na poprawienie błędów. Testy wykonywane manualnie oraz automatycznie za pomocą dostępnych framework'ów.

Testy integracyjne

Projekt oprogramowania składa się z wielu modułów, które są logicznie połączone ze sobą. Testy integracyjne koncentrują się na komunikacji danych między tymi modułami. Wykonywane manualnie.

Tabela 1 Scenariusze testów integracyjnych

ID	Cel	Opis	Oczekiwany rezultat
1	Sprawdzenie połączenia między oknem głównym strony, a listą wydarzeń.	Będąc na stronie głównej, kliknięcie przycisku od wyświetlenia wszystkich wydarzeń.	Wyświetlenie na stronie listy wydarzeń.
2	Sprawdzenie połączenia między listą wydarzeń, a stroną konkretnego wydarzenia.	Będąc na stronie z listą wydarzeń, kliknięcie przycisku od wyświetlenia opisu wydarzenia.	Wyświetlenie na stronie opisu wydarzenia.
3	Sprawdzenie połączenia między oknem głównym strony, a wyszukiwarką wydarzeń.	Będąc na stronie głównej, kliknięcie przycisku od wyszukiwania wydarzeń.	Wyświetlenie strony do wyszukiwania wydarzeń.
4	Sprawdzenie połączenia między strona wyszukiwania wydarzeń, a listą wyszukanych wydarzeń.	Będąc na stronie wyszukiwania wydarzeń, wpisanie słów kluczowych i kliknięcie przycisku do wyszukania wydarzeń.	Wyświetlenie strony z listą wydarzeń spełniających kryteria wyszukiwania.

Testy funkcjonalne

Testy weryfikujące wszystkie zakładane funkcje biznesowe aplikacji. Każda funkcjonalność serwisu jest testowana poprzez dostarczenie danych wejściowych oraz porównaniu rzeczywistych wyników z oczekiwanymi. Wykonywane są manualnie i nie dotyczą kodu źródłowego.

Tabela 2 Scenariusze testów funkcjonalnych

ID	Cel	Opis	Oczekiwany rezultat
1	Sprawdzenie listy nadchodzących wydarzeń.	Przejście do listy wydarzeń.	Wyświetlenie listy nadchodzących wydarzeń według domyślnych filtrów.
2	Wyszukanie listy wydarzeń.	Przejście do listy wydarzeń. Wprowadzenie słów kluczowych jako filtru wyszukiwania.	Walidacja wprowadzonych filtrów. Wyświetlenie listy wydarzeń spełniających kryteria wyszukiwania.

3	Sprawdzenie etapu dodawania wydarzeń	Przygotowanie wszystkich wymaganych oraz poprawnych danych nowego wydarzenia. Zalogowanie się jako administrator systemu. Przejście do strony dodawania wydarzeń, wpisanie danych i zaakceptowanie.	Walidacja wprowadzonych danych. Dodanie danych do bazy danych. Pozytywny komunikat z systemu. Wydarzenie znajduję się na liście wszystkich wydarzeń.
4	Sprawdzenie etapu dodawania wydarzeń	Przygotowanie niepełnych danych nowego wydarzenia. Zalogowanie się jako administrator systemu. Przejście do strony dodawania wydarzeń, wpisanie danych i zaakceptowanie.	Walidacja wprowadzonych danych. Negatywny komunikat z systemu.
5	Sprawdzenie etapu subskrypcji wydarzenia	Przejście do opisu nadchodzącego wydarzenia. Wpisanie danych do notyfikacji. Zasubskrybowanie wydarzenia.	Walidacja wprowadzonych danych. Dodanie danych subskrypcji do bazy. Pozytywny komunikat z systemu. Wyświetlenie powiadomienia o nadchodzącym wydarzeniu.

Testy wydajnościowe

Testy polegające na zmierzeniu wydajności serwisu pod dużym obciążeniem. Serwis internetowy może nie spełniać wymagań, jeśli w danym momencie będzie korzystało dużo użytkowników. Testy wykonywane manualnie oraz automatycznie z wykorzystaniem gotowych narzędzi jak np. Jmeter.

11. Wyniki testów

Testy jednostkowe backendu

Poniżej przedstawiamy wyniki testów jednostkowych backendu. Łączny czas wykonywania wynosi niewiele ponad 1 sekundę. Testy jednostkowe pokrywają takie funkcjonalności jak: dodawanie nowego wydarzenia, wyszukiwanie wydarzenia za pomocą daty wydarzenia, wyszukiwanie zaawansowane oraz pobieranie wszystkich wydarzeń. Testy odbywają się na usługach zaślepionych, tak aby weryfikacji podlegały jedynie funkcje odpowiadające za zwracanie danych z API. Weryfikacją połączenia ze źródłem danych zajmują się testy integracyjne.

6	0	0	1.010s
tests	failures	ignored	duration

100% successful

Tests | Standard output

Test	Duration	Result
addNewEvent_basicTest()	0.290s	passed
getEvent_findByDateStart()	0.265s	passed
getEvent_findByText()	0.044s	passed
getEvent_findByTextAndDates()	0.137s	passed
getEvents_basicTest()	0.187s	passed
getSpecificEvent_basicTest()	0.087s	passed

Dodatkowo załączamy kod źródłowy testów jednostkowych:

```
@Autowired
private WebApplicationContext wac;
private MockMvc mockMvc;

@BeforeEach
void setup() {
    this.mockMvc = MockMvcBuilders.webAppContextSetup(this.wac).build();
}

@Test
public void addNewEvent_basicTest() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.post( urlTemplate: "/events/add?logo=url&date=1900-01-01&event_category=Music&venue=Warszawa&summary=Opis&name=Koncert")).andExpect(MockMvcResultM
}

@Test
void getEvents_basicTest() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/events/all")).andExpect(MockMvcResultMatchers.status().isOk());
}

@Test
void getEvent_findByDateStart() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/events/search?dateStart=2019-01-01")).andExpect(MockMvcResultMatchers.status().isOk());
}

@Test
void getEvent_findByText() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/events/search?text=Ronstring")).andExpect(MockMvcResultMatchers.status().isOk());
}

@Test
void getEvent_findByTextAndDates() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/events/search?text=Ronstring&dateStart=2019-01-01&dateEnd=2999-01-01")).andExpect(MockMvcResultMatchers.status().isOk());
}

@Test
void getSpecificEvent_basicTest() throws Exception {
    mockMvc.perform(MockMvcRequestBuilders.get( urlTemplate: "/events/search/1")).andExpect(MockMvcResultMatchers.status().isOk());
}
```

Testy integracyjne backendu

Testy integracyjne mają za zadanie weryfikację ścieżki dodania nowego wydarzenia, a następnie wyszukanie go za pomocą różnych parametrów. Dzięki temu testy weryfikują połączenie ze źródłem danych, a także poprawność działania funkcji biznesowych.

2	0	0	0.427s
tests	failures	ignored	duration

100% successful

Tests | Standard output

Test	Duration	Result
findByNameContaining_returnEvent()	0.362s	passed
findByNameOrSummaryContainingOrDateIsBetween_returnEvent()	0.065s	passed

Poniżej kod źródłowy:

```java
@ExtendWith(SpringExtension.class)
@DataJpaTest
@AutoConfigureTestDatabase
class EventRepositoryTest {

    @Autowired
    private TestEntityManager entityManager;

    @Autowired
    private EventRepository eventRepository;

    @Test
    void findByNameOrSummaryContainingOrDateIsBetween_returnEvent() {
        LocalDate date = LocalDate.parse("2019-01-01");
        LocalDate startDate = date.plusDays(-7);
        LocalDate endDate = date.plusDays(7);
        Event testEvent = new Event( name: "test", summary: "testsummary", event_category: "testCategory", venue: "testVenue", date, logo: "testlogo");
        entityManager.persist(testEvent);
        entityManager.flush();

        List<Event> foundTestEvent = eventRepository.findByNameOrSummaryContainingOrDateIsBetween(testEvent.getName(), startDate, endDate);

        assertEquals(testEvent.getName(), foundTestEvent.get(0).getName());
    }

    @Test
    void findByNameContaining_returnEvent() {
        LocalDate date = LocalDate.parse("2019-01-01");
        Event testEvent = new Event( name: "test", summary: "testsummary", event_category: "testCategory", venue: "testVenue", date, logo: "testlogo");
        entityManager.persist(testEvent);
        entityManager.flush();

        List<Event> foundTestEvent = eventRepository.findByNameContaining(testEvent.getName());

        assertEquals(testEvent.getName(), foundTestEvent.get(0).getName());
    }
}
```