**Technologie użyte w projekcie**

Poniżej znajduje się kompletna dokumentacja zawierająca wszystkie informacje o wykorzystanych technologiach wraz z wersjami i kluczowymi konfiguracjami.

**1. Podsumowanie projektu**

* **Nazwa projektu:** Eventify
* **Cel:** Aplikacja do kupowania biletów na wydarzenia z możliwością dodawania wydarzeń do ulubionych z frontendem React/Vite oraz backendem Java Spring Boot.

**2. Środowisko programistyczne**

* **IDE/Edytor:**
  + Visual Studio Code 1.99.3 (user setup)
  + IntelliJ IDEA 2024.2.1 (Community Edition)
* **System operacyjny deweloperski:** Windows 11

**3. Frontend**

**3.1. Główne technologie**

* **React**
  + Wersja: ^19.0.0
  + Oficjalna dokumentacja: [https://reactjs.org](https://reactjs.org/)
* **Vite**
  + Wersja: ^6.3.3
  + Konfiguracja w vite.config.js:

import { defineConfig } from 'vite';

import react from '@vitejs/plugin-react';

export default defineConfig({

plugins: [react()],

server: {

proxy: {

'/api': 'http://localhost:8085'

}

}

});

* Projekt w czystym JavaScripcie

**3.2. Zarządzanie pakietami**

**Node.js:**

* + Wersja: v22.14.0
  + Pobierz z https://nodejs.org/

**npm:**

* + Wersja: 10.9.2
  + Instalowany razem z Node.js

**Skrypty w package.json:**

{

"scripts": {

"dev": "vite",

"build": "vite build",

"lint": "eslint .",

"preview": "vite preview"

}

}

**3.3. Zależności**

* **dependencies:**
  + bootstrap: ^5.3.5
  + react: ^19.0.0
  + react-dom: ^19.0.0
  + react-router-dom: ^7.5.2
* **devDependencies:**
  + @vitejs/plugin-react: ^4.3.4
  + vite: ^6.3.3
  + eslint: ^9.22.0 wraz z pluginami @eslint/js, eslint-plugin-react-hooks, eslint-plugin-react-refresh
  + Typowania: @types/react: ^19.0.10, @types/react-dom: ^19.0.4
  + globals: ^16.0.0

**3.4. Style i UI**

* **CSS framework:** Bootstrap 5
  + Wersja: 5.3.5
  + Importowane w plikach CSS/JS projektu
  + Import lokalny:

import 'bootstrap/dist/css/bootstrap.min.css';

**4. Backend**

**4.1. Język i runtime**

* **Java**
  + Wersja: 17 (JDK 17)

**4.2. Framework**

* **Spring Boot**
  + Wersja: 3.4.5
  + Parent POM: spring-boot-starter-parent:3.4.5

**4.3. Zależności w pom.xml**

<!-- Spring Boot Web + JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- Walidacja Bean -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-validation</artifactId>

</dependency>

<!-- Sterownik MySQL -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Devtools -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<!-- Bezpieczeństwo i JWT -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<!-- JAXB (XML Binding) -->

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>com.sun.xml.bind</groupId>

<artifactId>jaxb-impl</artifactId>

<version>2.3.5</version>

</dependency>

**4.4. Budowanie projektu**

* **Maven**
  + Wersja: Apache Maven 3.9.9
  + Maven home: C:\maven\apache-maven-3.9.9
  + Budowanie: mvn clean package

**4.5. Baza danych**

* **Rodzaj:** MySQL
* **URL:** jdbc:mysql://localhost:3306/eventify
* **Użytkownik:** root
* **Hasło:** skonfigurowane w application-local.properties
* **Dialect:** org.hibernate.dialect.MySQL8Dialect

Konfiguracja w src/main/resources/application.properties oraz application-local.properties:

# application.properties

spring.application.name=Eventify

spring.datasource.url=jdbc:mysql://localhost:3306/eventify

spring.datasource.username=root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.config.import=optional:application-local.properties

server.port=8085

# application-local.properties

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

jwt.secret=

jwt.expiration-ms=

**5. Inne narzędzia i usługi**

* **Kontrola wersji:** Git (repozytorium na GitHub)

**Dostępne endpointy wraz z typami danych**

Poniższa sekcja opisuje wszystkie dostępne endpointy w projekcie Eventify wraz z typami danych dla żądań (request) i odpowiedzi (response).

* 1. **Pobieranie wszystkich wydarzeń**

GET /api/events

**Opis:** Zwraca listę wszystkich wydarzeń.

**Request**

* Brak parametrów.

**Response**

* Status: 200 OK
* Body: tablica obiektów EventDto.

**Przykładowy payload**

[

{

"id": 1,

"title": "Koncert rockowy",

"description": "Energetyczny koncert rockowy na stadionie.",

"eventDate": "2025-05-10T18:00:00",

"createdAt": "2025-01-20T09:30:00",

"organizerId": 5,

"organizerName": "MusicFest Sp. z o.o.",

"cityId": 2,

"cityName": "Warszawa",

"street": "Marszałkowska",

"buildingNumber": "10",

"apartmentNumber": "5",

"postalCode": "00-001",

"price": 99.99

},

{

"id": 2,

"title": "Wystawa sztuki nowoczesnej",

"description": "Przegląd prac najciekawszych młodych artystów.",

"eventDate": "2025-05-12T12:00:00",

"createdAt": "2025-02-15T14:00:00",

"organizerId": 3,

"organizerName": "ArtGallery SA",

"cityId": 4,

"cityName": "Kraków",

"street": "Floriańska",

"buildingNumber": "30",

"apartmentNumber": null,

"postalCode": "31-019",

"price": 50.00

}

]

**Struktura EventDto:**

|  |  |  |
| --- | --- | --- |
| **Pole** | **Typ** | **Opis** |
| id | Long | Unikalny identyfikator wydarzenia. |
| title | String | Tytuł wydarzenia. |
| description | String | Opis wydarzenia. |
| eventDate | LocalDateTime | Data i czas wydarzenia (format ISO-8601, bez strefy czasowej). |
| createdAt | LocalDateTime | Data i czas utworzenia wpisu wydarzenia. |
| organizerId | Long | ID użytkownika (organizatora). |
| organizerName | String | Nazwa organizatora. |
| cityId | Long | ID miasta. |
| cityName | String | Nazwa miasta. |
| street | String | Ulica lokalizacji wydarzenia. |
| buildingNumber | String | Numer budynku. |
| apartmentNumber | String (nullable) | Numer lokalu (opcjonalnie). |
| postalCode | String | Kod pocztowy. |
| price | BigDecimal | Cena biletu w walucie PLN. |

* 1. **Pobieranie polubionych wydarzeń**

GET /api/events/liked

**Opis:** Zwraca listę wydarzeń, które zostały przez aktualnie zalogowanego użytkownika oznaczone jako „ulubione”. Wymagana jest autoryzacja (token JWT).

**Request**

* Authorization: Token JWT w nagłówku.

**Response**

* Status: 200 OK
* Body: tablica obiektów EventDto reprezentujących ulubione wydarzenia.

**Przykładowy payload**

[

{

"id": 42,

"title": "Koncert Jazzowy",

"description": "Wieczór pełen najlepszych standardów jazzowych.",

"eventDate": "2025-06-15T19:30:00",

"createdAt": "2024-11-01T14:20:35",

"organizerId": 7,

"organizerName": "Jazz Club Warsaw",

"cityId": 2,

"cityName": "Warszawa",

"street": "Marszałkowska",

"buildingNumber": "10",

"apartmentNumber": null,

"postalCode": "00-624",

"price": 120.00

},

{

"id": 57,

"title": "Warsztaty Ceramiki",

"description": "Zajęcia z tworzenia unikatowych naczyń.",

"eventDate": "2025-05-20T10:00:00",

"createdAt": "2025-02-10T09:15:00",

"organizerId": 12,

"organizerName": "ArtCraft",

"cityId": 3,

"cityName": "Kraków",

"street": "Floriańska",

"buildingNumber": "5A",

"apartmentNumber": "2",

"postalCode": "31-019",

"price": 75.50

}

]

**Struktura EventDto** (struktura zamieszczona w endpoincie GET /api/events)

* 1. **Przesyła polubione wydarzenia**

POST /api/events/{eventId}/like

**Opis:** Oznacza dane wydarzenie jako „ulubione” dla aktualnie zalogowanego użytkownika. Wymagana jest autoryzacja (token JWT).

**Request**

* Path Parameter: eventId identyfikator wydarzenia, które użytkownik dodaje do ulubionych.
* Authorization: Token JWT w nagłówku.

**Response**

* Status:
  + 200 OK: Sukces – relacja „użytkownik polubił wydarzenie” zapisana.
  + 204 No Content: Sukces – wydarzenie usunięte z ulubionych.
  + 401 Unauthorized: Brak lub nieważny token.
  + 404 Not Found: Relacja „ulubione” nie istnieje (niepolubione).
  + 500 Internal Error: Inny błąd po stronie serwera.

**Przykładowy respone, gdy wydarzenie jest już polubione:**

{

"timestamp": "2025-05-06T13:45:22.987+00:00",

"status": 400,

"error": "Bad Request",

"message": "Użytkownik o id=7 już polubił wydarzenie o id=42",

"path": "/api/events/42/like"

}

* 1. **Usuwa polubionych wydarzeń**

DELETE /api/events/{id}/liked

**Opis:** Usuwa relację „ulubione wydarzenie” dla zalogowanego użytkownika i wydarzenia o podanym id. Wymagana jest autoryzacja (token JWT).

**Request**

* Path Parameter: id identyfikator wydarzenia, które ma zostać usunięte z ulubionych.
* Authorization: Token JWT w nagłówku.

**Response**

* Status:
  + 204 No Content: Sukces – wydarzenie usunięte z ulubionych.
  + 401 Unauthorized: Brak lub nieważny token.
  + 404 Not Found: Relacja „ulubione” nie istnieje (niepolubione).
  + 500 Internal Error: Inny błąd po stronie serwera.

**Przykładowe payload:**

* 204 No Content: brak odpowiedzi.
* Błąd 404:

{

"timestamp": "2025-05-06T15:42:10.123",

"status": 404,

"error": "Not Found",

"message": "Like not found with userId,eventId=[userId=5,eventId=123]",

"path": "/api/events/123/liked"

}

* 1. **Pobiera zakupione wydarzenia**

GET /api/events/bought\_events

**Opis:** Zwraca listę wszystkich wydarzeń, na które zalogowany użytkownik zakupił bilety.

**Request**

* Authorization: Token JWT zalogowanego uzytkownika.

**Response**

* Status:
  + 200 OK: Sukces – wydarzenie usunięte z ulubionych.
  + 401 Unauthorized: Brak lub nieważny token.
  + 500 Internal Server Error: Inny błąd po stronie serwera.

**Przykładowy payload**

[

{

"id": 101,

"title": "Festiwal Filmowy",

"description": "Projekcje najlepszych filmów europejskich.",

"eventDate": "2025-07-10T18:00:00",

"createdAt": "2025-01-15T12:30:00",

"organizerId": 3,

"organizerName": "CinemaClub",

"cityId": 5,

"cityName": "Gdańsk",

"street": "Długa",

"buildingNumber": "15",

"apartmentNumber": null,

"postalCode": "80-831",

"price": 50.00,

"amount": 2,

"priceAll": 100.00

},

{

"id": 202,

"title": "Koncert Rockowy",

"description": "Największe hity lat 80-tych na żywo.",

"eventDate": "2025-05-20T20:00:00",

"createdAt": "2024-12-05T09:45:00",

"organizerId": 8,

"organizerName": "LiveMusic",

"cityId": 1,

"cityName": "Warszawa",

"street": "Nowy Świat",

"buildingNumber": "20",

"apartmentNumber": "3",

"postalCode": "00-029",

"price": 120.00,

"amount": 4,

"priceAll": 480.00

}

]

**Struktura EventDto + pola BoughtEvent:**

| **Pole** | **Typ** | **Opis** |
| --- | --- | --- |
| id | Long | Unikalny identyfikator wydarzenia. |
| title | String | Tytuł wydarzenia. |
| description | String | Opis wydarzenia. |
| eventDate | String | Data i godzina rozpoczęcia. |
| createdAt | String | Data utworzenia wydarzenia w systemie. |
| organizerId | Long | ID organizatora. |
| organizerName | String | Nazwa organizatora. |
| cityId | Long | ID miasta. |
| cityName | String | Nazwa miasta. |
| street | String | Ulica. |
| buildingNumber | String | Numer budynku. |
| apartmentNumber | String | null | Numer mieszkania (opcjonalnie). |
| postalCode | String | Kod pocztowy. |
| price | Number | Cena pojedynczego biletu. |
| amount | Integer | Liczba zakupionych biletów (z tabeli bought\_events). |
| priceAll | Number | Łączna kwota za wszystkie bilety (z bought\_events). |

* 1. **Przesyła zakupione bilety**

POST /api/events/{eventId}/buy

**Opis:** Pozwala zalogowanemu użytkownikowi zakupić bilety na wybrane wydarzenie. Przekazywane są szczegóły zakupu takie jak data wydarzenia, liczba biletów oraz ich łączna cena.

**Request**

* Path Parameter: eventId ID wydarzenia, na które kupowany jest bilet.
* Authorization: Token JWT zalogowanego użytkownika.

**Response**

* Status:
  + 201 Created: Sukces – bilet(y) zapisane w bazie (BoughtEvent); brak ciała.
  + 400 Bad Request: Niepoprawne lub brakujące pola w ciele żądania.
  + 401 Unauthorized: Brak lub nieważny token.
  + 404 Not Found: Wydarzenie o podanym eventId nie istnieje.
  + 500 Internal Error: Inny błąd po stronie serwera.
* Body: tablica obiektów BoughtEventRequestDto.

**Przykładowy payload:**

{

"eventDate": "2025-06-15T19:30:00",

"amount": 2,

"priceAll": 240.00

}

**Struktura BoughtEventRequestDto:**

| **Pole** | **Typ** | **Opis** |
| --- | --- | --- |
| eventDate | String | Data i godzina wydarzenia (kopiowana z EventDto). |
| amount | int | Liczba kupowanych biletów. |
| priceAll | double | Łączna kwota za wszystkie bilety. |

* 1. **Pobiera wszystkie miasta**

GET /api/cities

**Opis:** Zwraca listę wszystkich miast zapisanych w bazie danych.

**Request**

* Brak parametrów

**Response**

* Status: 200 OK

**Przykładowy payload:**

[

{

"id": 1,

"name": "Warszawa",

"state": "Mazowieckie",

"country": "Polska",

"createdAt": "2024-10-12T08:45:30.123"

},

{

"id": 2,

"name": "Kraków",

"state": "Małopolskie",

"country": "Polska",

"createdAt": "2024-11-05T14:20:10.456"

}

]

**Struktura modelu City:**

| **Pole** | **Typ** | **Opis** |
| --- | --- | --- |
| id | Long | Unikalny identyfikator miasta. |
| name | String | Nazwa miasta (niepusta). |
| state | String lub null | Województwo/prowincja (opcjonalnie). |
| country | String | Kraj (niepuste). |
| createdAt | String | Data i czas utworzenia rekordu (ustawiane automatycznie). |

* 1. **Przesyłanie danych istniejącego użytkownika**

POST /api/auth/login

**Opis:** Autoryzacja użytkownika: sprawdza dane logowania (email + hasło), a w przypadku powodzenia zwraca token JWT.

**Request**

* Brak parametrów.

**Response**

• Status:

* + 200 OK: Zalogowano pomyślnie. W ciele token i dane użytkownika.
  + 400 Bad Request: Nie spełniono walidacji (np. niepoprawny email lub za krótkie hasło).
  + 401 Unauthorized: Nieprawidłowy email lub hasło.
  + 500 Internal Error: Inny błąd serwera.

• Body: tablica obiektów LoginRequestDto.

**Przykładowy payload:**

{

"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJqYW4ua293YWxza2lAZXhhbXBsZS5jb20iLCJpYXQiOjE2ODI1NzIwMDB9.XYZabcd1234EfGhIjKLmnOpQrStUvWxYz",

"userId": 15,

"userName": "Jan Kowalski"

}

**Struktura LoginRequestDto:**

| **Pole** | **Typ** | **Walidacja** |
| --- | --- | --- |
| email | String | @Email, niepuste |
| password | String | min. 8 znaków, niepuste |

**Struktura JwtRespone:**

| **Pole** | **Typ** | **Opis** |
| --- | --- | --- |
| token | String | JWT do autoryzacji kolejnych żądań |
| userId | Long | ID zalogowanego użytkownika |
| userName | String | Nazwa użytkownika (imię i nazwisko) |

* 1. **Przesyłanie danych nowego użytkownika**

POST /api/auth/register

**Opis:** Rejestruje nowego użytkownika w systemie. Waliduje dane wejściowe (imię, email, hasło), zapisuje użytkownika z zahashowanym hasłem i zwraca dane nowo utworzonego konta.

**Request**

• Brak parametrów.

**Response**

• Status:

* + 201 Created: Rejestracja zakończona sukcesem. Ciało odpowiedzi zawiera dane nowego użytkownika.
  + 400 Bad Request: Niepowodzenie walidacji pól (np. niepoprawny e-mail, za krótkie hasło, brak imienia).
  + 409 Conflict: Istnieje już użytkownik z podanym adresem e-mail (ResourceAlreadyExistsException).
  + 500 Internal Error: Inny, nieoczekiwany błąd po stronie serwera.

• Body: tablica obiektów UserDto.

**Przykładowy payload:**

{

"id": 23,

"name": "Anna Nowak",

"email": "anna.nowak@example.com",

"createdAt": "2025-05-06T12:34:56Z"

}

**Struktura RegisterRequestDto:**

| **Pole** | **Typ** | **Walidacja** |
| --- | --- | --- |
| name | String | niepuste |
| email | String | @Email, niepuste, unikalny |
| password | String | @NotBlank, @Size(min=8), musi zawierać cyfrę |

**Struktura UserDto:**

| **Pole** | **Typ** | **Opis** |
| --- | --- | --- |
| id | Long | Unikalny identyfikator nowego użytkownika |
| name | String | Imię i nazwisko użytkownika |
| email | String | Adres e-mail użytkownika |
| createdAt | String | Data i godzina utworzenia konta |

**Wkład członków zespołu w rozwój aplikacji**

**Nina Łopacińska -** głównie odpowiedzialna za rozwój warstwy frontendowej aplikacji oraz części funkcjonalności po stronie backendu związanych z interfejsem użytkownika i obsługą wydarzeń. Główne zadania na kolejnych etapach projektu:

* Stworzenie struktury projektu po stronie frontendowej i backendowej
* Zaciągnięcie bootstrapa
* Implementacja komponentów takich jak: EventsList.
* Wprowadzenie filtrowania wydarzeń.
* Dodanie przycisków i funkcjonalności zakupu biletów oraz polubień wydarzeń.
* Poprawa wyglądu i estetyki frontendu.
* Dodanie kontrolera miast (CityController) w backendzie.
* Implementacja i modyfikacja cen biletów.
* Refaktoryzacja oraz poprawki istniejących widoków i komponentów frontendowych.
* Czyszczenie frontendu, wyłączenie wspólnych elementów

**Paweł Żurawski -** rozwój backendu aplikacji w technologii Java/Spring Boot oraz integracji backendu z frontendem. Główne zadania na kolejnych etapach projektu:

* „Postawienie” działającego projektu - stworzenie współdziałającego frontendu i backendu oraz bazy danych (pierwszej, testowej wersji aplikacji), stworzenie repozytorium.
* Wstępna implementacja wszystkich klas backendu
* Implementacja logiki rejestracji i logowania użytkowników, wraz z refaktoryzacją klas związanych z autoryzacją.
* Stworzenie odpowiednich zestawów zakładek dla użytkowników i gości
* Zablokowanie dostępu do kupna biletów / lajkowania wydarzeń dla gości
* Implementacja funkcjonalności lajkowania i odlajkowywania wydarzeń.
* Dodanie obsługi wylogowania
* Zabezpieczenie widoku „/MyEvents” przed dostępem dla niezalogowanych użytkowników.
* Czyszczenie i refaktoryzacja kodu backendowego