

Dokumentacja Techniczna – Menedżer Wydarzeń

1. Szczegółowy Opis Aplikacji

1.1. Wprowadzenie

Aplikacja "Menedżer Wydarzeń" to system webowy zaprojektowany w architekturze klient-serwer, którego głównym celem jest umożliwienie użytkownikom zarządzania wydarzeniami. Składa się z dwóch głównych komponentów: aplikacji frontendowej działającej w przeglądarce użytkownika oraz backendowego interfejsu API obsługującego logikę biznesową i komunikację z bazą danych. Aplikacja pozwala na rejestrację, logowanie użytkowników oraz wykonywanie operacji CRUD (Create, Read, Update, Delete) na wydarzeniach. Została stworzona z myślą o osobach lub grupach potrzebujących prostego narzędzia do organizacji i śledzenia planowanych spotkań, konferencji czy innych aktywności.

1.2. Architektura Systemu

System opiera się na klasycznej architekturze trójwarstwowej:

- 1. Warstwa Prezentacji (Frontend):** Interfejs użytkownika zbudowany przy użyciu HTML, CSS oraz JavaScript (z wykorzystaniem frameworka Bootstrap 5 do stylizacji i komponentów). Komunikuje się z backendem poprzez żądania HTTP (asynchroniczne, np. Fetch API lub XMLHttpRequest) do zdefiniowanych endpointów API. Odpowiada za wyświetlanie danych, obsługę interakcji użytkownika (formularze, przyciski) oraz nawigację między widokami (routing po stronie klienta).
- 2. Warstwa Aplikacji (Backend):** Zbudowana w oparciu o Node.js z frameworkiem Express. Implementuje logikę biznesową aplikacji, w tym:
 - Obsługę żądań HTTP przychodzących od frontendu.
 - Walidację danych wejściowych (np. przy rejestracji, tworzeniu/edycji wydarzeń) za pomocą express-validator.
 - Uwierzytelnianie i autoryzację użytkowników przy użyciu tokenów JWT (jsonwebtoken).
 - Komunikację z warstwą danych w celu zapisu i odczytu informacji.
 - Definiowanie i udostępnianie RESTful API, którego dokumentacja może być generowana i przeglądana za pomocą swagger-jsdoc i swagger-ui-express.
- 3. Warstwa Danych (Baza Danych):** Wykorzystuje relacyjną bazę danych PostgreSQL do przechowywania informacji o użytkownikach i wydarzeniach. Interakcja z bazą danych od strony backendu odbywa się za pomocą biblioteki knex, która pełni rolę Query Buildera, ułatwiając tworzenie i wykonywanie zapytań

SQL oraz zarządzanie migracjami schematu bazy danych. Baza danych jest zarządzana i uruchamiana w kontenerze Docker, co zapewnia spójne środowisko niezależnie od systemu operacyjnego dewelopera (docker-compose.yml).

1.3. Stos Technologiczny

Backend:

Środowisko uruchomieniowe: Node.js

Framework: Express 5.x

Baza Danych: PostgreSQL (>= 17.4, zarządzana przez Docker)

ORM/Query Builder: Knex.js

Sterownik Bazy Danych: pg (node-postgres)

Uwierzytelnianie: JSON Web Tokens (JWT) (jsonwebtoken)

Haszowanie haseł: bcrypt (bcrypt)

Walidacja: express-validator

Obsługa CORS: cors

Zmienne Środowiskowe: dotenv

Generowanie ID: integer

Dokumentacja API: Swagger (swagger-jsdoc, swagger-ui-express)

Narzędzia deweloperskie: Nodemon (do automatycznego restartu serwera), Concurrently (do równoległego uruchamiania skryptów)

Testowanie: Jest (jest), Supertest (supertest) - zgodnie z package.json i wymaganiami PDF.

Frontend:

Struktura: HTML5

Stylizacja: CSS3, Bootstrap 5 (bootstrap, bootstrap-icons)

Logika: JavaScript (ES6 Modules, Vanilla JS)

Narzędzia deweloperskie: Live Server (do podglądu na żywo), Concurrently

Konteneryzacja:

Docker & Docker Compose (dla bazy danych PostgreSQL)

System Kontroli Wersji: Git

1.4. Główne Funkcjonalności

Aplikacja realizuje podstawowe funkcje systemu zarządzania wydarzeniami, w tym:

1. Zarządzanie Użytkownikami:

Rejestracja: Umożliwia nowym użytkownikom tworzenie konta poprzez podanie danych (np. email, hasło). Hasła są haszowane przed zapisem do bazy danych za pomocą bcrypt.

Logowanie: Zarejestrowani użytkownicy mogą zalogować się do systemu. Po pomyślnym uwierzytelnieniu backend generuje token JWT, który jest przesyłany do frontendu.

Autoryzacja: Token JWT jest wykorzystywany do autoryzacji dostępu do chronionych zasobów API (np. tworzenie, edycja, usuwanie wydarzeń). Frontend przesyła token w nagłówku Authorization przy każdym chronionym żądaniu.

2. Zarządzanie Wydarzeniami (Operacje CRUD): Aplikacja implementuje pełen cykl zarządzania wydarzeniami:

Tworzenie (Create): Zalogowani administratorzy mogą dodawać nowe wydarzenia poprzez formularz (zawierający pola takie jak: nazwa, data, godzina, lokalizacja, opis). Dane są walidowane po stronie backendu.

Odczyt (Read): Użytkownicy mogą przeglądać listę dostępnych wydarzeń oraz szczegóły pojedynczego wydarzenia. Dostęp do listy może być publiczny, podczas gdy szczegóły lub akcje mogą wymagać zalogowania.

Aktualizacja (Update): Zalogowani administratorzy mogą modyfikować dane istniejących wydarzeń.

Usuwanie (Delete): Zalogowani administratorzy mogą usuwać wydarzenia.

3. Interfejs Użytkownika:

Nawigacja: Aplikacja posiada pasek nawigacyjny (#navbar-container) renderowany dynamicznie przez JS, umożliwiający poruszanie się między różnymi sekcjami (np. Strona główna, Lista wydarzeń, Moje wydarzenia, Logowanie/Rejestracja).

Routing: Frontend implementuje mechanizm routingu po stronie klienta, dynamicznie wstrzykując odpowiednie widoki do kontenera #app-content bez konieczności przeładowywania całej strony.

Responsywność: Interfejs jest zaprojektowany z myślą o responsywności (wymagane minimum 2 breakpointy wg PDF), dostosowując się do różnych rozmiarów ekranów dzięki wykorzystaniu siatki i komponentów Bootstrap.

Interaktywność: Formularze, przyciski, powiadomienia (np. o sukcesie/błędzie operacji) są obsługiwane przez JavaScript. Wyświetlany jest wskaźnik ładowania (spinner-border) podczas oczekiwania na odpowiedź z API.

Przechowywanie Stanu/Danych: Aplikacja wykorzystuje localStorage do przechowywania np. tokenu JWT lub innych danych sesji użytkownika po stronie klienta.

4. API Backendowe:

Udostępnia RESTful API z jasno zdefiniowanymi endpointami dla operacji na użytkownikach i wydarzeniach.

Stosuje odpowiednie metody HTTP (GET, POST, PUT, DELETE).

Zwraca poprawne kody statusu HTTP (np. 200, 201, 204, 400, 401, 403, 404, 409).

Zabezpiecza endpointy wymagające uwierzytelnienia za pomocą middleware sprawdzającego ważność tokenu JWT.

Umożliwia konfigurację CORS (cors) w celu zezwolenia na żądania z domeny frontendowej.

Posiada zintegrowaną dokumentację Swagger dostępną pod dedykowanym endpointem (np. /api-docs).

1.5. Baza Danych

System: PostgreSQL.

Interakcja: Poprzez bibliotekę Knex.js, która abstrahuje zapytania SQL.

1.6. Bezpieczeństwo

Hasła użytkowników są bezpiecznie przechowywane w bazie danych dzięki haszowaniu bcrypt.

Sesje użytkowników są zarządzane za pomocą tokenów JWT, co minimalizuje ryzyko ataków typu CSRF (jeśli tokeny są przechowywane np. w localStorage).

Dostęp do chronionych operacji API jest ograniczony tylko dla zalogowanych i autoryzowanych użytkowników.

Podstawowa walidacja danych wejściowych po stronie backendu (express-validator) zapobiega prostym atakom i zapewnia integralność danych.

Konfiguracja CORS zapobiega nieautoryzowanym żądaniom z innych domen.

1.7. Uruchomienie i Testowanie

Uruchomienie: Baza danych jest uruchamiana za pomocą docker compose. Serwer backendowy uruchamiany jest poleceniem npm. Frontend jest serwowany za pomocą npm. Całość można uruchomić jednym poleceniem npm run dev w katalogu backendu dzięki concurrently.

Testowanie: Projekt zawiera zestaw wszystkich możliwych scenariuszy testowych dla każdego endpointu w całym systemie - łącznie 375 testów.

2. Funkcjonalności Aplikacji (User Stories)

Poniżej przedstawiono kluczowe funkcjonalności aplikacji z perspektywy różnych typów użytkowników:

1. Rejestracja użytkownika:

Jako niezarejestrowany użytkownik,

chcę móc utworzyć nowe konto w systemie podając swój adres e-mail i hasło,

aby uzyskać dostęp do funkcji tworzenia i zarządzania własnymi biletami.

2. Logowanie użytkownika:

Jako zarejestrowany użytkownik,

chcę móc zalogować się do aplikacji przy użyciu mojego adresu e-mail i hasła,

aby uzyskać dostęp do spersonalizowanych treści i możliwości zarządzania moimi biletami.

3. Przeglądanie listy wydarzeń (Publiczne):

Jako dowolny użytkownik (zarejestrowany lub nie),

chcę móc przeglądać listę wszystkich publicznie dostępnych wydarzeń (np. ich nazwy i daty),

aby zorientować się, jakie wydarzenia są planowane.

4. Tworzenie nowego wydarzenia:

Jako zalogowany administrator,

chcę móc dodać nowe wydarzenie poprzez formularz, podając jego nazwę, opis, datę, godzinę i lokalizację,

aby zaplanować i opublikować własne spotkanie lub aktywność.

5. Przeglądanie szczegółów wydarzenia:

Jako zalogowany użytkownik,

chcę móc wyświetlić szczegółowe informacje o wybranym wydarzeniu (pełen opis, dokładna data, lokalizacja),

aby uzyskać wszystkie potrzebne informacje na jego temat.

6. Edycja własnego wydarzenia:

Jako zalogowany administrator,

chcę móc edytować szczegóły wydarzenia, które wcześniej stworzyłem,

aby zaktualizować informacje, poprawić błędy lub zmienić plan (np. datę, miejsce).

7. Usuwanie własnego wydarzenia:

Jako zalogowany administrator,
chcę móc usunąć wydarzenie, które wcześniej stworzyłem,
aby odwołać wydarzenie lub usunąć je z systemu, gdy nie jest już aktualne.

8. Wylogowanie z systemu:

Jako zalogowany użytkownik,
chcę móc się wylogować z aplikacji,
aby bezpiecznie zakończyć sesję i chronić swoje konto przed nieautoryzowanym dostępem.

9. Przeglądanie użytkowników:

Jako zalogowany administrator,
chcę mieć dedykowaną sekcję, aby zobaczyć użytkowników z ich rolami,
aby łatwo zarządzać użytkownikami.

10. Walidacja danych w formularzach:

Jako użytkownik (rejestrujący się lub tworzący/edytujący wydarzenie),
chcę otrzymywać informacje o błędach w formularzu (np. brakujące pole, niepoprawny format e-mail),
aby wiedzieć, co należy poprawić, aby pomyślnie przesłać dane.

3. Odbiorcy Systemu

Aplikacja "Menedżer Wydarzeń" jest przeznaczona dla różnych grup użytkowników i interesariuszy, którzy mogą korzystać z jej funkcjonalności lub być zaangażowani w jej działanie. Główni odbiorcy systemu to:

1. Organizatorzy Wydarzeń (np. Liderzy Zespołów, Koordynatorzy Projektów, Osoby Prywatne):

Cel: Główni użytkownicy aplikacji, odpowiedzialni za tworzenie, aktualizowanie i usuwanie wydarzeń. Wykorzystują system do planowania, informowania innych i zarządzania logistyką spotkań, warsztatów, konferencji czy innych aktywności grupowych lub prywatnych.

Interakcja: Korzystają z pełnego zakresu funkcji CRUD dla wydarzeń oraz funkcji zarządzania kontem (rejestracja, logowanie).

2. Uczestnicy / Osoby Zainteresowane (np. Członkowie Zespołów, Goście, Społeczność):

Cel: Użytkownicy (zarówno zalogowani, jak i anonimowi), którzy chcą być na bieżąco z planowanymi wydarzeniami. Przeglądają listę wydarzeń i ich szczegóły, aby uzyskać informacje o terminach, miejscach i tematyce.

Interakcja: Głównie odczytują dane – przeglądają listę wydarzeń i ich szczegóły. Mogą (jeśli są zarejestrowani) korzystać z funkcji logowania dla potencjalnie spersonalizowanych widoków lub przyszłych funkcji (np. zapisy na wydarzenia).

3. Administratorzy Systemu:

Cel: Osoby odpowiedzialne za techniczne utrzymanie aplikacji. Monitorują działanie serwera, zarządzają bazą danych (backupy, optymalizacja), dbają o bezpieczeństwo i dostępność systemu. Mogą mieć również uprawnienia do zarządzania kontami użytkowników (np. resetowanie haseł, blokowanie kont) w bardziej rozbudowanych wersjach.

Interakcja: Bezpośrednia interakcja z infrastrukturą serwerową (Node.js, Docker, PostgreSQL), logami systemowymi, potencjalnie z panelem administracyjnym (jeśli zostanie dodany).

4. Deweloperzy Aplikacji:

Cel: Zespół lub osoba odpowiedzialna za tworzenie, rozwijanie i naprawianie błędów w aplikacji. Pracują z kodem źródłowym (backend i frontend), konfiguracją środowiska, systemem kontroli wersji i narzędziami deweloperskimi.

Interakcja: Bezpośrednia praca z kodem, narzędziami (Node.js, npm, Docker, Git), bazą danych (schemat, migracje), testami (Jest, Supertest) oraz dokumentacją API (Swagger).

4. Potencjalne Korzyści dla Użytkowników

Korzystanie z aplikacji "Menedżer Wydarzeń" przynosi szereg korzyści zarówno dla organizatorów, jak i uczestników wydarzeń:

1. **Centralizacja Informacji:** Wszystkie dane dotyczące planowanych wydarzeń (terminy, miejsca, opisy) są zgromadzone w jednym, łatwo dostępnym miejscu, eliminując potrzebę przeszukiwania e-maili, komunikatorów czy oddzielnych kalendarzy.
2. **Usprawnione Planowanie:** Organizatorzy otrzymują dedykowane narzędzie, które ułatwia proces tworzenia i definiowania szczegółów wydarzeń w ustrukturyzowany sposób.
3. **Łatwość Aktualizacji Danych:** W przypadku zmian (np. daty, lokalizacji), organizator może szybko zaktualizować informacje o wydarzeniu, a zmiany są natychmiast widoczne dla wszystkich zainteresowanych.

4. **Poprawa Komunikacji:** System działa jak jedno, spójne źródło informacji (Single Source of Truth), co minimalizuje ryzyko nieporozumień i błędnych interpretacji dotyczących szczegółów wydarzenia.
5. **Oszczędność Czasu dla Organizatorów:** Automatyzacja procesu zarządzania wydarzeniami (tworzenie, edycja, usuwanie) jest znacznie szybsza niż ręczne informowanie uczestników czy zarządzanie danymi w arkuszach kalkulacyjnych.
6. **Oszczędność Czasu dla Uczestników:** Uczestnicy mogą samodzielnie i szybko sprawdzić wszystkie potrzebne informacje o wydarzeniu w dowolnym momencie, bez konieczności kontaktowania się z organizatorem.
7. **Lepsza Organizacja:** Systematyczne zarządzanie listą wydarzeń (bieżących i przyszłych) pozwala na lepsze planowanie i unikanie konfliktów terminów.
8. **Stały Dostęp do Informacji:** Aplikacja jest dostępna przez przeglądarkę internetową, co umożliwia dostęp do informacji o wydarzeniach z różnych urządzeń (komputer, tablet, smartfon) i lokalizacji.
9. **Przejrzystość Danych:** Standaryzowany sposób prezentacji informacji o wydarzeniach (nazwa, data, opis, lokalizacja) sprawia, że są one czytelne i łatwe do zrozumienia dla wszystkich użytkowników.
10. **Archiwizacja Wydarzeń:** System przechowuje informacje o zakończonych wydarzeniach, co może być przydatne do celów historycznych, raportowania lub planowania przyszłych, podobnych aktywności.
11. **Podstawa do Rozbudowy:** Istniejąca struktura aplikacji ułatwia dodawanie w przyszłości bardziej zaawansowanych funkcji, takich jak system powiadomień, możliwość zapisywania się na wydarzenia czy integracja z zewnętrznymi kalendarzami, co dodatkowo zwiększy wartość dla użytkowników.

5. Opis Modułów Aplikacji

Aplikacja "Menedżer Wydarzeń" jest podzielona na logiczne moduły, z których każdy odpowiada za określony obszar funkcjonalności i zarządzanie powiązanymi danymi. Poniżej opisano kluczowe moduły (istniejące i potencjalne):

1. Moduł Authentication (Uwierzytelnianie):

Cel: Zapewnienie bezpiecznego dostępu do aplikacji i ochrona zasobów.

Odpowiedzialność: Obsługa procesów rejestracji nowych użytkowników, logowania istniejących użytkowników oraz wylogowywania. Zarządza generowaniem, weryfikacją i obsługą tokenów (np. JWT) używanych do autoryzacji żądań. Odpowiada również za bezpieczne przechowywanie i porównywanie haseł (np. z użyciem bcrypt).

Główne Funkcjonalności: Rejestracja, Logowanie, Wylogowanie, Generowanie tokenu JWT, Weryfikacja tokenu JWT (middleware).

Powiązane Dane: Głównie operuje na danych z modułu User, przechowuje klucze/secrety do JWT.

Powiązania: Ściśle powiązany z modulem User. Chroni dostęp do endpointów w innych modułach (np. Event, Ticket).

2. Moduł User (Użytkownik):

Cel: Zarządzanie informacjami o użytkownikach systemu.

Odpowiedzialność: Przechowywanie danych użytkowników (e-mail, hasło - hashowane, ewentualnie inne dane profilowe). Umożliwia operacje CRUD na kontach użytkowników (choć w podstawowej wersji może to być ograniczone do tworzenia przy rejestracji i odczytu przez system). Może w przyszłości zarządzać rolami i uprawnieniami.

Główne Funkcjonalności: Tworzenie użytkownika (rejestracja), Odczyt danych użytkownika (np. do weryfikacji logowania, powiązania z wydarzeniem), Aktualizacja profilu (potencjalna), Usunięcie konta (potencjalna).

Powiązane Dane: Encja User (tabela users).

Powiązania: Authentication (dostarcza dane do logowania/rejestracji), Event (jako twórca wydarzenia), Ticket (jako posiadacz biletu/uczestnik).

3. Moduł Event (Wydarzenie):

Cel: Centralny moduł do zarządzania wydarzeniami.

Odpowiedzialność: Umożliwia tworzenie, odczytywanie, aktualizowanie i usuwanie (CRUD) wydarzeń. Przechowuje wszystkie kluczowe informacje o wydarzeniu, takie jak nazwa, opis, data, lokalizacja (tekstowa lub jako powiązanie), twórca.

Główne Funkcjonalności: Dodawanie nowego wydarzenia, Pobieranie listy wydarzeń, Pobieranie szczegółów jednego wydarzenia, Modyfikacja istniejącego wydarzenia, Usunięcie wydarzenia.

Powiązane Dane: Encja Event (tabela events).

Powiązania: Locale (miejsce), Category (kategoria), Prelegent (prelegent), Ticket (bilet), Catering (katering), Resource (sprzęt), Sponsor (sponsor).

4. Moduł Prelegent (Mówca/Prezenter):

Cel: Zarządzanie informacjami o osobach prowadzących prelekcje lub warsztaty podczas wydarzeń.

Odpowiedzialność: Przechowywanie danych prelegentów (nazwa, opis) oraz zarządzanie ich powiązaniami z konkretnymi wydarzeniami lub sesjami w ramach wydarzeń.

Główne Funkcjonalności: Dodawanie/Usuwanie prelegenta, Edycja danych prelegenta, Przypisywanie prelegenta do wydarzenia/sesji, Listowanie prelegentów dla danego wydarzenia.

Powiązane Dane: Encja Event_Prelegent.

Powiązania: Event, User (prelegent może być użytkownikiem systemu).

5. Moduł Locale (Miejsce/Lokalizacja):

Cel: Zarządzanie szczegółowymi informacjami o miejscach, w których odbywają się wydarzenia.

Odpowiedzialność: Przechowywanie danych o lokalizacjach (miasto, nazwa). Umożliwia powiązanie wydarzeń z konkretnymi, predefiniowanymi miejscami.

Główne Funkcjonalności: CRUD dla lokalizacji, Przypisywanie lokalizacji do wydarzenia, Wyszukiwanie/Filtrowanie lokalizacji.

Powiązane Dane: Encja Locale.

Powiązania: Event.

6. Moduł Category (Kategoria):

Cel: Klasyfikacja wydarzeń w celu ułatwienia ich przeglądania i filtrowania.

Odpowiedzialność: Zarządzanie kategoriami wydarzeń (np. Konferencja, Warsztat, Networking, Koncert, Sport). Umożliwia przypisanie jednej lub wielu kategorii do wydarzenia.

Główne Funkcjonalności: CRUD dla kategorii, Przypisywanie kategorii do wydarzenia, Filtrowanie wydarzeń po kategorii.

Powiązane Dane: Encja Event_Category.

Powiązania: Event.

7. Moduł Catering (Obsługa gastronomiczna):

Cel: Zarządzanie opcjami cateringowymi dostępnymi na wydarzeniach.

Odpowiedzialność: Przechowywanie informacji o dostawcach cateringu, menu, opcjach dietetycznych. Umożliwia powiązanie wybranych opcji cateringowych z wydarzeniem.

Główne Funkcjonalności: CRUD dla opcji cateringowych/dostawców, Przypisywanie cateringu do wydarzenia, Zarządzanie preferencjami żywieniowymi uczestników (w powiązaniu z Ticket lub User).

Powiązane Dane: Encja Event_Catering.

Powiązania: Event.

8. Moduł Resource (Zasób):

Cel: Zarządzanie zasobami niezbędnymi do organizacji wydarzeń (np. sale, sprzęt AV, materiały).

Odpowiedzialność: Przechowywanie informacji o dostępnych zasobach, zarządzanie ich rezerwacjami i dostępnością w kontekście planowanych wydarzeń.

Główne Funkcjonalności: CRUD dla zasobów, Rezerwacja zasobu dla wydarzenia, Sprawdzanie dostępności zasobu.

Powiązane Dane: Encja Event_Resource.

Powiązania: Event.

9. Moduł Ticket (Bilet/Rejestracja):

Cel: Obsługa procesu rejestracji uczestników na wydarzenia i zarządzanie biletami.

Odpowiedzialność: Definiowanie typów biletów (np. darmowe, płatne, VIP), zarządzanie ich ilością, obsługa procesu "zakupu".

Główne Funkcjonalności: Definiowanie typów biletów dla wydarzenia, Rejestracja użytkownika na wydarzenie (powiązanie User-Event poprzez Ticket), Sprawdzanie statusu biletu/rejestracji, Listowanie uczestników wydarzenia.

Powiązane Dane: Encja Event_Ticket.

Powiązania: Event, User.

10. Moduł Sponsor (Sponsor):

Cel: Zarządzanie informacjami o sponsorach wspierających wydarzenia.

Odpowiedzialność: Przechowywanie danych sponsorów (nazwa, opis) oraz zarządzanie ich powiązaniami z konkretnymi wydarzeniami.

Główne Funkcjonalności: CRUD dla sponsorów, Przypisywanie sponsora do wydarzenia, Wyświetlanie listy sponsorów na stronie wydarzenia.

Powiązane Dane: Encja Event_Sponsor.

Powiązania: Event.

6. Strategia Testowania Endpointów API

Aplikacja "Menedżer Wydarzeń" kładzie nacisk na niezawodność i poprawność działania backendowego API. W tym celu stosowana jest strategia kompleksowego testowania endpointów, wykorzystująca framework **Jest** oraz bibliotekę **Supertest**. Testy są uruchamiane w dedykowanym środowisku (NODE_ENV=test), które korzysta z osobnej bazy danych, inicjalizowanej i czyszczonej przed/po uruchomieniu zestawu testów, co gwarantuje izolację i powtarzalność.

Każdy istotny endpoint API podlega rygorystycznym testom obejmującym szeroki wachlarz scenariuszy, aby zapewnić jego solidność:

6.1. Zakres Testowanych Scenariuszy dla Endpointów:

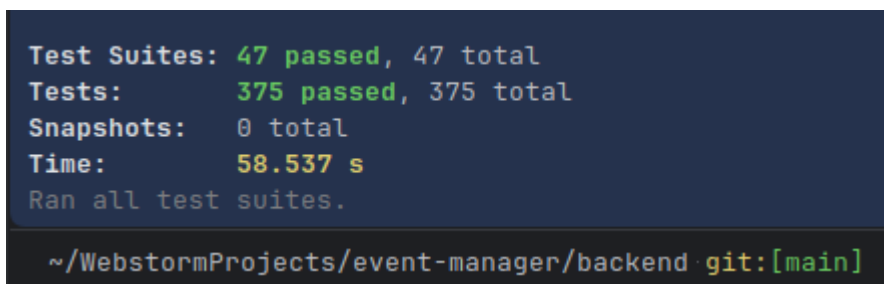
Dla każdego endpointu (np. POST /api/users/register, POST /api/users/login, GET /api/events, POST /api/events, PUT /api/events/:id, DELETE /api/events/:id) weryfikowane są następujące kategorie przypadków:

1. Przypadki Pozytywne (Happy Paths):

Poprawność Działania: Weryfikacja, czy endpoint realizuje swoje podstawowe zadanie przy poprawnych danych wejściowych i w odpowiednim stanie systemu (np. czy POST /api/events poprawnie tworzy wydarzenie w bazie danych i zwraca status 201 Created wraz z danymi nowo utworzonego zasobu).

Zgodność Odpowiedzi: Sprawdzenie, czy struktura i zawartość odpowiedzi (ciało JSON) są zgodne z oczekiwaniami (np. czy GET /api/events/:id zwraca wszystkie oczekiwane pola wydarzenia).

Poprawne Kody Statusu HTTP: Weryfikacja, czy zwracane są odpowiednie kody sukcesu (np. 200 OK, 201 Created, 204 No Content).



```
Test Suites: 47 passed, 47 total
Tests:      375 passed, 375 total
Snapshots:  0 total
Time:       58.537 s
Ran all test suites.

~/WebstormProjects/event-manager/backend git:[main]
```

2. Walidacja Danych Wejściowych (Input Validation):

Wykorzystanie express-validator jest testowane pod kątem reguł zdefiniowanych dla każdego endpointu.

Brakujące Pola: Testowanie reakcji endpointu, gdy brakuje wymaganych pól w ciele żądania (payload).

Niepoprawne Typy Danych: Testowanie, gdy dostarczone dane mają nieprawidłowy typ (np. string zamiast liczby, obiekt zamiast stringa).

Niepoprawne Formaty: Weryfikacja obsługi niepoprawnych formatów (np. nieprawidłowy adres e-mail, nieprawidłowy format daty).

Niespełnione Ograniczenia: Testowanie wartości poza dozwolonym zakresem (np. zbyt krótki/długi string, wartość numeryczna poza limitem).

Oczekiwany Rezultat: Weryfikacja zwrócenia kodu statusu 400 Bad Request (lub 422 Unprocessable Entity) wraz z czytelnym komunikatem o błędach walidacji.

3. Uwierzytelnianie i Autoryzacja (Authentication & Authorization):

Dostęp do Endpointów Publicznych: Weryfikacja, czy endpointy oznaczone jako publiczne są dostępne bez tokenu uwierzytelniającego.

Dostęp do Endpointów Chronionych:

Brak Tokenu: Testowanie reakcji (oczekiwany status 401 Unauthorized) przy próbie dostępu bez przesłania tokenu JWT.

Niepoprawny/Nieważny Token: Testowanie reakcji (oczekiwany status 401 Unauthorized) przy użyciu sfałszowanego, nieprawidłowo podpisanego lub wygasłego tokenu.

Niewystarczające Uprawnienia (oczekiwany status 403 Forbidden), gdy użytkownik z poprawnym tokenem próbuje wykonać akcję, do której nie ma uprawnień (np. edycja wydarzenia).

Poprawny Token: Weryfikacja, czy dostęp jest przyznawany przy użyciu ważnego tokenu JWT.

4. Obsługa Błędów i Przypadki Brzegowe (Error Handling & Edge Cases):

Zasób Nie Znaleziony: Testowanie reakcji endpointów operujących na konkretnym zasobie (np. GET /api/events/:id, PUT /api/events/:id, DELETE /api/events/:id), gdy podany identyfikator (:id) nie istnieje w bazie danych (oczekiwany status 404 Not Found).

Konflikty Zasobów: Testowanie scenariuszy, gdzie próba utworzenia zasobu narusza unikalne ograniczenia (np. rejestracja użytkownika z już istniejącym adresem e-mail - oczekiwany status 409 Conflict).

Błędy Wewnętrzne Serwera: Chociaż trudniejsze do bezpośredniego testowania, sprawdzana jest ogólna odporność aplikacji na nieoczekiwane błędy (np. poprzez mockowanie błędów bazy danych) i weryfikacja zwracania statusu 500 Internal Server Error w takich sytuacjach, bez ujawniania wrażliwych informacji.

Puste Wartości/Dane Graniczne: Testowanie z użyciem pustych stringów, wartości null, 0, maksymalnych dopuszczalnych długości stringów itp., aby upewnić się, że logika aplikacji poprawnie je obsługuje.

6.2. Przykład Strategii dla Endpointu POST /api/events:

Happy Path:

Test: Poprawne utworzenie wydarzenia z wszystkimi wymaganymi i opcjonalnymi polami przez zalogowanego użytkownika.

Weryfikacja: Status 201, poprawna struktura odpowiedzi, zapis w bazie danych.

Walidacja:

Test: Próba utworzenia wydarzenia z brakującą nazwą.

Weryfikacja: Status 400, komunikat błędu o braku nazwy.

Test: Próba utworzenia wydarzenia z datą w niepoprawnym formacie.

Weryfikacja: Status 400, komunikat błędu o formacie daty.

Autoryzacja:

Test: Próba utworzenia wydarzenia bez tokenu JWT.

Weryfikacja: Status 401.

Test: Próba utworzenia wydarzenia z nieważnym tokenem JWT.

Weryfikacja: Status 401.

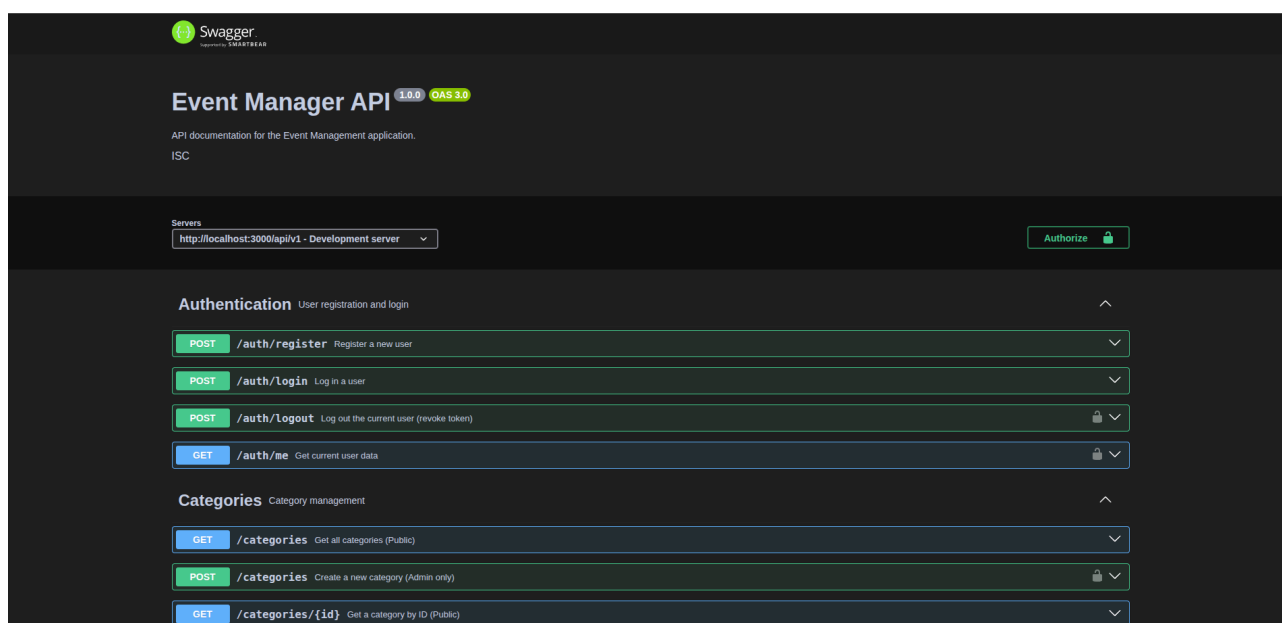
Edge Cases:

Test: Próba utworzenia wydarzenia z bardzo długim opisem (jeśli jest limit).

Weryfikacja: Status 400 lub poprawne obcięcie/obsługa.

7. Dokumentacja API (Swagger/OpenAPI)

Integralną częścią backendu aplikacji "Menedżer Wydarzeń" jest interaktywna dokumentacja API, generowana i udostępniana zgodnie ze standardem **OpenAPI**



Specification (formalnie znanym jako Swagger Specification).

7.1. Implementacja:

Generowanie Specyfikacji: Dokumentacja jest generowana dynamicznie na podstawie adnotacji **JSDoc** umieszczonych bezpośrednio w kodzie źródłowym, nad definicjami tras (endpointów) i schematów danych (modeli). Wykorzystywana jest do tego biblioteka `swagger-jsdoc`. Pozwala to na utrzymanie dokumentacji blisko kodu, którego dotyczy, co zwiększa jej aktualność i spójność.

Deweloperzy opisują każdy endpoint, określając jego ścieżkę, metodę HTTP (GET, POST, PUT, DELETE itp.), oczekiwane parametry (ścieżki, zapytania, nagłówka), strukturę ciała żądania (request body) oraz możliwe odpowiedzi (responses) wraz z ich kodami statusu i schematami danych.

Definiowane są również schematy (schemas) dla modeli danych (np. User, Event, ErrorResponse) oraz wymagania dotyczące bezpieczeństwa (np. konieczność przesłania tokenu JWT).

Udostępnianie Dokumentacji: Wygenerowana specyfikacja OpenAPI jest następnie serwowana jako interaktywny interfejs użytkownika za pomocą biblioteki `swagger-ui-express`. Tworzy ona dedykowany endpoint w aplikacji backendowej, pod którym dostępna jest przyjazna dla użytkownika, wizualna reprezentacja API.

7.2. Dostęp:

Interaktywna dokumentacja Swagger UI jest **dostępna w przeglądarce internetowej po uruchomieniu serwera backendowego** (np. za pomocą `npm run dev` lub `npm start`). Zazwyczaj jest ona hostowana pod standardowym adresem URL, takim jak:

`http://localhost:3000/api-docs`

7.3. Korzyści z Dokumentacji Swagger:

Integracja Swagger/OpenAPI przynosi wiele korzyści:

1. **Interaktywność:** Umożliwia przeglądanie wszystkich dostępnych endpointów, ich parametrów, schematów żądań i odpowiedzi bezpośrednio w przeglądarce.
2. **Możliwość Testowania:** Funkcja "Try it out" w Swagger UI pozwala na wysyłanie rzeczywistych żądań do API bezpośrednio z poziomu dokumentacji, co jest niezwykle pomocne podczas dewelopmentu i testowania.
3. **Jasność i Standardyzacja:** Dostarcza jednoznacznego, ustandaryzowanego opisu kontraktu API, eliminując niejasności dotyczące sposobu interakcji z backendem.
4. **Ułatwiona Współpraca:** Stanowi "jedno źródło prawdy" (Single Source of Truth) dla deweloperów frontendowych i backendowych (oraz innych konsumentów API), usprawniając komunikację i integrację.
5. **Automatyzacja:** Dokumentacja jest generowana częściowo automatycznie z kodu, co zmniejsza nakład pracy związany z jej tworzeniem i utrzymaniem w porównaniu do pisania jej całkowicie ręcznie.

6. **Wsparcie dla Narzędzi:** Specyfikacja OpenAPI może być wykorzystana przez różne narzędzia do automatycznego generowania kodu klienta (SDK), testów czy dalszej walidacji.

8. Instalacja i Uruchomienie Projektu

8.1. Wymagania wstępne:

Przed przystąpieniem do instalacji, upewnij się, że na Twoim komputerze zainstalowane jest następujące oprogramowanie:

- **Node.js:** Dostępne na: <https://nodejs.org/>
- **Docker i Docker Compose:** Narzędzia do konteneryzacji, niezbędne do uruchomienia bazy danych. Dostępne na: <https://www.docker.com/get-started>
- **Git:** System kontroli wersji, potrzebny do pobrania kodu źródłowego aplikacji. Dostępne na: <https://git-scm.com/>

8.2. Instalacja i Uruchomienie:

Poniższe kroki należy wykonać w terminalu lub wierszu poleceń, zaczynając od głównego katalogu sklonowanego repozytorium projektu.

1. **Przejdź do katalogu frontendu:**
`cd frontend`
2. **Zainstaluj zależności frontendu:**
`npm install`
3. **Przejdź do katalogu backendu:**
`cd ../backend`
4. **Skonfiguruj zmienne środowiskowe backendu:**
`cp .env.example .env`
5. **Uruchom kontener z bazą danych:**
`docker compose up -d`
6. **Zainstaluj zależności backendu:**
`npm install`
7. **Wykonaj migracje bazy danych:**
`npx knex migrate:latest`
8. **Wypełnij bazę danych początkowymi danymi (opcjonalne, ale zalecane do testów):**
`npx knex seed:run`
9. **Uruchom aplikację w trybie deweloperskim:**
`npm run dev`
10. **Uruchom testy backendu (opcjonalne):**
`npm test`

8.3. Dostęp do aplikacji:

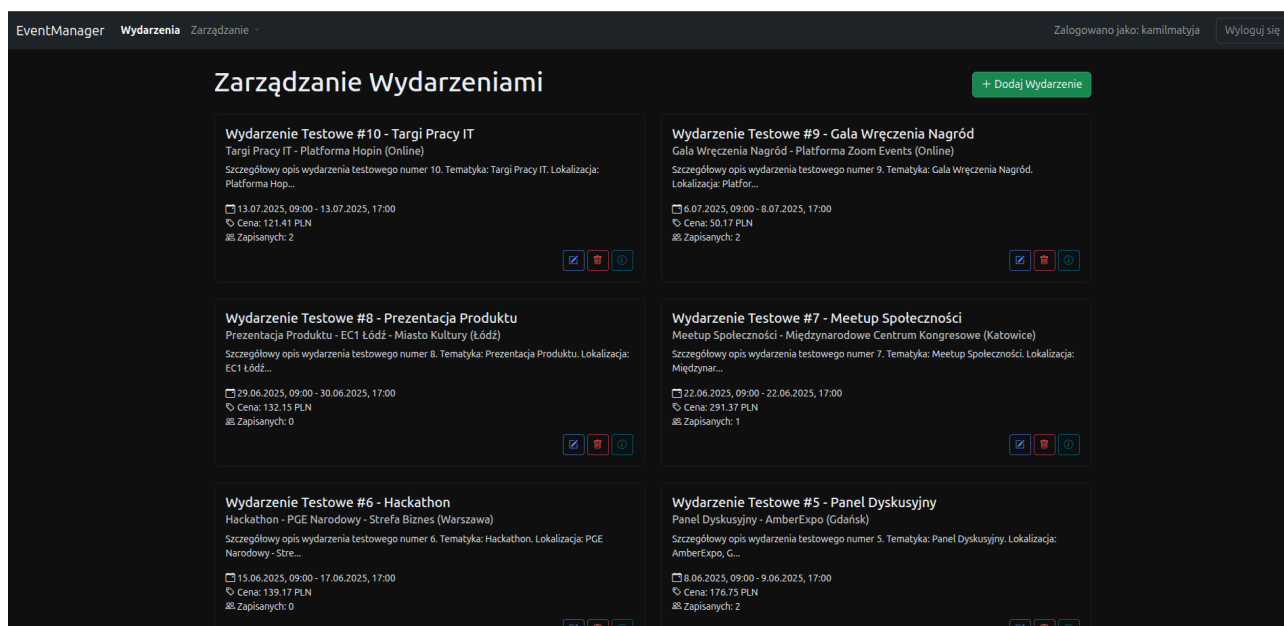
Po poprawnym wykonaniu kroków instalacji i uruchomienia poszczególne części aplikacji powinny być dostępne pod następującymi adresami w przeglądarce internetowej:

- **Interfejs użytkownika (Frontend):** `http://localhost:8080`
- **API Backendowe:** `http://localhost:3000`

- Dokumentacja API (Swagger UI): <http://localhost:3000/api-docs>

9. Responsywność Interfejsu Użytkownika (Frontend)

Interfejs użytkownika aplikacji "Menedżer Wydarzeń" został zaprojektowany i zaimplementowany z myślą o responsywności (Responsive Web Design - RWD). Celem jest zapewnienie optymalnego doświadczenia użytkownika (UX) oraz czytelności i użyteczności aplikacji na szerokiej gamie urządzeń i rozmiarów ekranów, od dużych



monitorów desktopowych po tablety i smartfony.

Zarządzanie Wydarzeniami

+ Dodaj Wydarzenie

Wydarzenie Testowe #10 - Targi Pracy IT
Targi Pracy IT - Platforma Hopin (Online)
Szczegółowy opis wydarzenia testowego numer 10. Tematyka: Targi Pracy IT. Lokalizacja: Platforma Hop...

13.07.2025, 09:00 - 13.07.2025, 17:00

Cena: 121.41 PLN

Zapisanych: 2

✎

✖

⌵

Wydarzenie Testowe #9 - Gala Wręczenia Nagród
Gala Wręczenia Nagród - Platforma Zoom Events (Online)
Szczegółowy opis wydarzenia testowego numer 9. Tematyka: Gala Wręczenia Nagród. Lokalizacja: Platfor...

6.07.2025, 09:00 - 8.07.2025, 17:00

Cena: 50.17 PLN

Zapisanych: 2

✎

✖

⌵

Wydarzenie Testowe #8 - Prezentacja Produktu
Prezentacja Produktu - EC1 Łódź - Miasto Kultury (Łódź)
Szczegółowy opis wydarzenia testowego numer 8. Tematyka: Prezentacja Produktu. Lokalizacja: EC1 Łódź...

29.06.2025, 09:00 - 30.06.2025, 17:00

Cena: 132.15 PLN

Zapisanych: 0

✎

✖

⌵

Wydarzenie Testowe #7 - Meetup Społeczności
Meetup Społeczności - Międzynarodowe Centrum Kongresowe (Katowice)
Szczegółowy opis wydarzenia testowego numer 7. Tematyka: Meetup Społeczności. Lokalizacja: Międzynar...

22.06.2025, 09:00 - 22.06.2025, 17:00

Cena: 291.37 PLN

Zapisanych: 1

✎

✖

⌵

Wydarzenie Testowe #6 - Hackathon
Hackathon - PGE Narodowy - Strefa Biznes (Warszawa)
Szczegółowy opis wydarzenia testowego numer 6. Tematyka: Hackathon. Lokalizacja: PGE Narodowy - Stre...

15.06.2025, 09:00 - 17.06.2025, 17:00

Cena: 139.17 PLN

Zapisanych: 0

✎

✖

⌵

Wydarzenie Testowe #5 - Panel Dyskusyjny
Panel Dyskusyjny - AmberExpo (Gdańsk)
Szczegółowy opis wydarzenia testowego numer 5. Tematyka: Panel Dyskusyjny. Lokalizacja: AmberExpo, G...

8.06.2025, 09:00 - 9.06.2025, 17:00

Cena: 176.75 PLN

Zapisanych: 2

✎

✖

⌵

Zarządzanie Wydarzeniami

+ Dodaj Wydarzenie

Wydarzenie Testowe #10 - Targi Pracy IT
Targi Pracy IT - Platforma Hopin (Online)
Szczegółowy opis wydarzenia testowego numer 10. Tematyka: Targi Pracy IT. Lokalizacja: Platforma Hop...

13.07.2025, 09:00 - 13.07.2025, 17:00

Cena: 121.41 PLN

Zapisanych: 2

✎

✖

⌵

Wydarzenie Testowe #9 - Gala Wręczenia Nagród
Gala Wręczenia Nagród - Platforma Zoom Events (Online)
Szczegółowy opis wydarzenia testowego numer 9. Tematyka: Gala Wręczenia Nagród. Lokalizacja: Platfor...

6.07.2025, 09:00 - 8.07.2025, 17:00

Cena: 50.17 PLN

Zapisanych: 2

✎

✖

⌵

Wydarzenie Testowe #8 - Prezentacja Produktu
Prezentacja Produktu - EC1 Łódź - Miasto Kultury (Łódź)
Szczegółowy opis wydarzenia testowego numer 8. Tematyka: Prezentacja Produktu. Lokalizacja: EC1 Łódź...

29.06.2025, 09:00 - 30.06.2025, 17:00

Cena: 132.15 PLN

Zapisanych: 0

✎

✖

⌵

Wydarzenie Testowe #7 - Meetup Społeczności
Meetup Społeczności - Międzynarodowe Centrum Kongresowe (Katowice)
Szczegółowy opis wydarzenia testowego numer 7. Tematyka: Meetup Społeczności. Lokalizacja: Międzynar...

22.06.2025, 09:00 - 22.06.2025, 17:00

Cena: 291.37 PLN

Zapisanych: 1

✎

✖

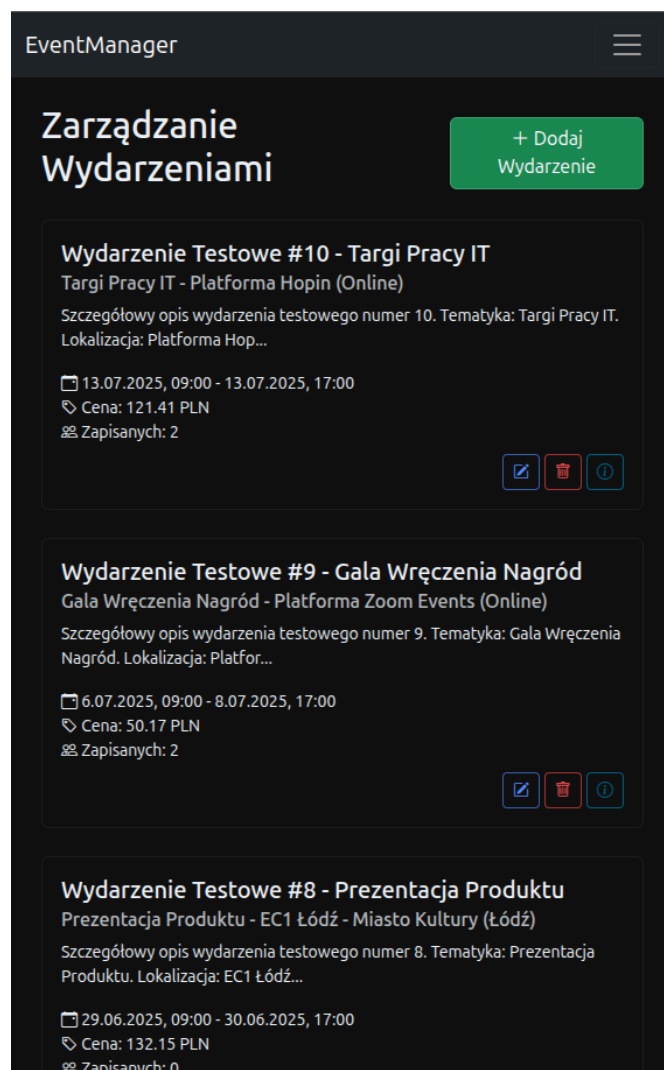
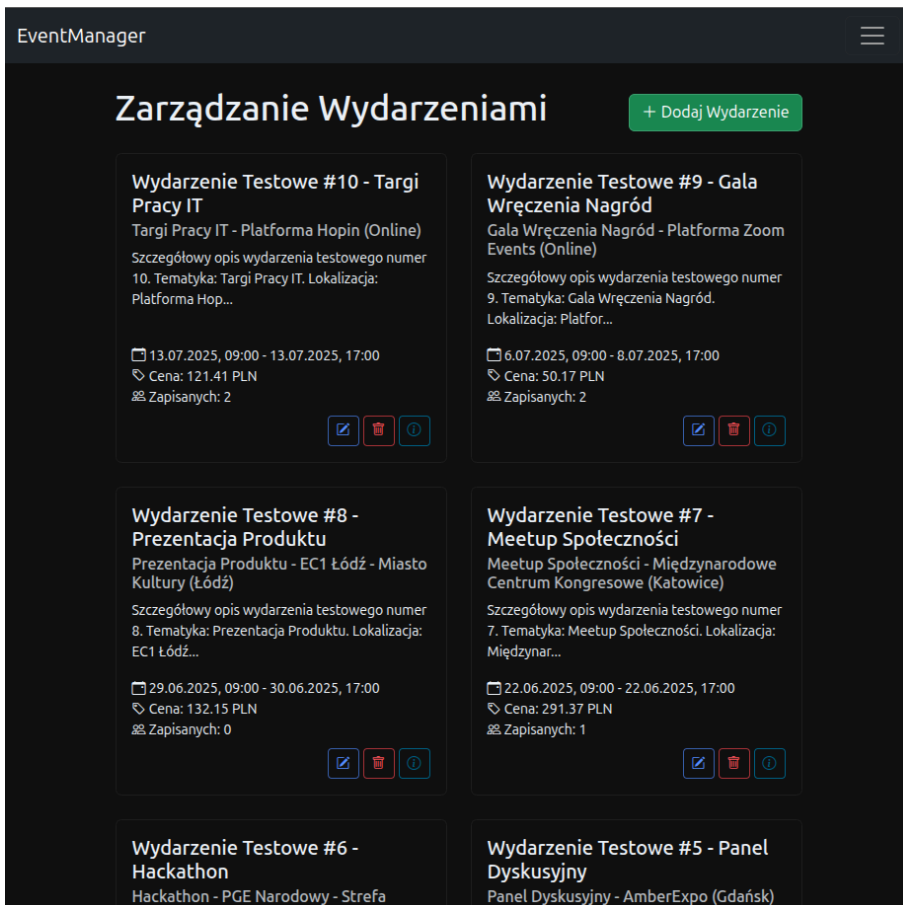
⌵

Wydarzenie Testowe #6 - Hackathon
Hackathon - PGE Narodowy - Strefa Biznes (Warszawa)
Szczegółowy opis wydarzenia testowego numer 6. Tematyka: Hackathon. Lokalizacja: PGE Narodowy - Stre...

15.06.2025, 09:00 - 17.06.2025, 17:00

Wydarzenie Testowe #5 - Panel Dyskusyjny
Panel Dyskusyjny - AmberExpo (Gdańsk)
Szczegółowy opis wydarzenia testowego numer 5. Tematyka: Panel Dyskusyjny. Lokalizacja: AmberExpo, G...

8.06.2025, 09:00 - 9.06.2025, 17:00



9.1. Podstawy Technologiczne:

Responsywność frontendu jest osiągnięta przede wszystkim dzięki wykorzystaniu frameworka **Bootstrap 5**. Bootstrap jest frameworkiem typu "mobile-first", co oznacza, że jego komponenty i system siatki są domyślnie zoptymalizowane pod kątem urządzeń mobilnych, a następnie skalują się w górę dla większych ekranów.

9.2. Kluczowe Mechanizmy Implementacji:

1. **Meta Tag Viewport:** W pliku frontend/index.html znajduje się kluczowy meta tag:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Instruuje on przeglądarki mobilne, aby ustawiły szerokość obszaru widzenia (viewport) na szerokość ekranu urządzenia i zastosowały początkowy poziom powiększenia 1.0. Jest to fundamentalny krok do włączenia responsywnego zachowania.

2. **System Siatki Bootstrap (Grid System):** Chociaż konkretne widoki są wstrzykiwane dynamicznie przez JavaScript, zakładaną praktyką przy budowie interfejsu z Bootstrapem jest wykorzystanie jego potężnego, 12-kolumnowego systemu siatki (.container, .row, .col-*). Pozwala on na:

Definiowanie układu treści w wierszach i kolumnach.

Określenie, jak kolumny mają się układać i zmieniać szerokość na różnych predefiniowanych **punktach przerwania** (breakpoints: sm, md, lg, xl, xxl), które odpowiadają typowym szerokościom ekranów urządzeń.

Automatyczne "łamanie" kolumn (układanie jedna pod drugą) na mniejszych ekranach, co zapobiega konieczności poziomego przewijania.

3. **Komponenty Responsywne Bootstrap:** Wiele standardowych komponentów Bootstrap, takich jak pasek nawigacyjny (Navbar), modale (Modal), karty (Card) czy formularze (Form), posiada wbudowane zachowania responsywne. Przykładowo:

Pasek nawigacyjny automatycznie zwija swoje elementy do przycisku typu "hamburger" na mniejszych ekranach.

Tabele mogą być opakowane w responsywne kontenery, aby umożliwić przewijanie poziome tylko dla tabeli, gdy nie mieści się na ekranie.

4. **Klasy Użytkowe Bootstrap (Utility Classes):** Bootstrap dostarcza wielu klas użytkowych do kontrolowania widoczności (.d-none, .d-md-block), wyrównania (.text-center, .text-lg-start), marginesów i paddingów (.m-1, .p-sm-2) w sposób responsywny, co pozwala na precyzyjne dostosowanie wyglądu na różnych ekranach.

9.3. Zgodność z Wymaganiami:

Projekt spełnia wymagania określone w dokumencie zaliczeniowym, zapewniając responsywność kluczowych widoków aplikacji **dla co najmniej pięciu różnych punktów przerwania (breakpointów)**. Oznacza to, że układ i prezentacja treści adaptują się

poprawnie np. przy przejściu z widoku desktopowego (lg lub xl) na widok tabletowy (md) i mobilny (sm).

9.4. Weryfikacja:

Responsywność aplikacji można łatwo zweryfikować za pomocą:

Narzędzi deweloperskich przeglądarki: Większość nowoczesnych przeglądarek (Chrome, Firefox, Edge) oferuje tryb "Responsive Design Mode" lub "Device Simulation", który pozwala na podgląd strony przy różnych predefiniowanych lub niestandardowych rozdzielczościach ekranu.

Zmiany rozmiaru okna przeglądarki: Ręczne zmniejszanie i zwiększanie szerokości okna przeglądarki na komputerze desktopowym również pokaże, jak interfejs adaptuje się do zmian.

10. Lista Endpointów API

Poniżej znajduje się lista endpointów API udostępnianych przez backend aplikacji "Menedżer Wydarzeń", wraz z krótkim opisem ich funkcjonalności.

Authentication (Uwierzytelnianie)

POST /auth/register - Rejestruje nowego użytkownika w systemie.

POST /auth/login - Loguje zarejestrowanego użytkownika i zwraca token uwierzytelniający.

POST /auth/logout - Wylogowuje aktualnie uwierzytelnionego użytkownika, potencjalnie unieważniając jego token.

GET /auth/me - Pobiera dane aktualnie uwierzytelnionego użytkownika.

Categories (Kategorie)

GET /categories - Pobiera listę wszystkich dostępnych kategorii wydarzeń.

POST /categories - Tworzy nową kategorię wydarzeń (wymaga uprawnień administratora).

GET /categories/{id} - Pobiera szczegóły konkretnej kategorii na podstawie jej ID.

PUT /categories/{id} - Aktualizuje dane istniejącej kategorii o podanym ID (wymaga uprawnień administratora).

DELETE /categories/{id} - Usuwa kategorię o podanym ID (wymaga uprawnień administratora).

Catering (Obsługa gastronomiczna)

GET /caterings - Pobiera listę wszystkich zdefiniowanych opcji/dostawców cateringu.

POST /caterings - Tworzy nową opcję lub dostawcę cateringu (wymaga uprawnień administratora).

GET /caterings/{id} - Pobiera szczegóły konkretnej opcji/dostawcy cateringu na podstawie ID.

PUT /caterings/{id} - Aktualizuje dane istniejącej opcji/dostawcy cateringu o podanym ID (wymaga uprawnień administratora).

DELETE /caterings/{id} - Usuwa opcję/dostawcę cateringu o podanym ID (wymaga uprawnień administratora).

Events (Wydarzenia)

GET /events - Pobiera listę wszystkich wydarzeń.

POST /events - Tworzy nowe wydarzenie (wymaga uprawnień administratora lub zalogowanego użytkownika, zależnie od implementacji).

GET /events/{id} - Pobiera szczegóły konkretnego wydarzenia na podstawie jego ID.

PUT /events/{id} - Aktualizuje dane istniejącego wydarzenia o podanym ID (wymaga odpowiednich uprawnień).

DELETE /events/{id} - Usuwa wydarzenie o podanym ID (wymaga odpowiednich uprawnień).

Tickets (Bilety/Rejestracje)

GET /tickets/my - Pobiera listę biletów/rejestracji należących do aktualnie zalogowanego użytkownika.

POST /tickets - Rejestruje zalogowanego użytkownika na wydarzenie (tworzy bilet/rejestrację).

DELETE /tickets/{id} - Anuluje rejestrację na wydarzenie poprzez usunięcie biletu o podanym ID (wymaga bycia właścicielem biletu).

Locales (Miejsca/Lokalizacje)

GET /locales - Pobiera listę wszystkich zdefiniowanych miejsc/lokalizacji wydarzeń.

POST /locales - Tworzy nowe miejsce/lokalizację (wymaga uprawnień administratora).

GET /locales/{id} - Pobiera szczegóły konkretnego miejsca/lokalizacji na podstawie ID.

PUT /locales/{id} - Aktualizuje dane istniejącego miejsca/lokalizacji o podanym ID (wymaga uprawnień administratora).

DELETE /locales/{id} - Usuwa miejsce/lokalizację o podanym ID (wymaga uprawnień administratora).

Prelegents (Prelegenci)

GET /prelegents - Pobiera listę wszystkich zdefiniowanych prelegentów/mówców.

POST /prelegents - Tworzy nowy profil prelegenta/mówcy (wymaga uprawnień administratora).

GET /prelegents/{id} - Pobiera szczegóły konkretnego prelegenta/mówcy na podstawie ID.

PUT /prelegents/{id} - Aktualizuje dane istniejącego prelegenta/mówcy o podanym ID (wymaga uprawnień administratora).

DELETE /prelegents/{id} - Usuwa profil prelegenta/mówcy o podanym ID (wymaga uprawnień administratora).

Resources (Zasoby)

GET /resources - Pobiera listę wszystkich zdefiniowanych zasobów (np. sale, sprzęt).

POST /resources - Tworzy nową definicję zasobu (wymaga uprawnień administratora).

GET /resources/{id} - Pobiera szczegóły konkretnego zasobu na podstawie ID.

PUT /resources/{id} - Aktualizuje dane istniejącego zasobu o podanym ID (wymaga uprawnień administratora).

DELETE /resources/{id} - Usuwa definicję zasobu o podanym ID (wymaga uprawnień administratora).

Sponsors (Sponsorzy)

GET /sponsors - Pobiera listę wszystkich zdefiniowanych sponsorów.

POST /sponsors - Tworzy nowy profil sponsora (wymaga uprawnień administratora).

GET /sponsors/{id} - Pobiera szczegóły konkretnego sponsora na podstawie ID.

PUT /sponsors/{id} - Aktualizuje dane istniejącego sponsora o podanym ID (wymaga uprawnień administratora).

DELETE /sponsors/{id} - Usuwa profil sponsora o podanym ID (wymaga uprawnień administratora).

Users (Użytkownicy)

GET /users - Pobiera listę wszystkich zarejestrowanych użytkowników (wymaga uprawnień administratora).

POST /users - Tworzy nowego użytkownika (wymaga uprawnień administratora).

GET /users/{id} - Pobiera szczegóły konkretnego użytkownika na podstawie jego ID (wymaga uprawnień administratora).

PUT /users/{id} - Aktualizuje dane istniejącego użytkownika o podanym ID (wymaga uprawnień administratora).

DELETE /users/{id} - Usuwa użytkownika o podanym ID (wymaga uprawnień administratora).

11. Struktura Bazy Danych (Tabele)

Poniżej opisano tabele tworzące schemat bazy danych aplikacji "Menedżer Wydarzeń", wraz z ich przeznaczeniem i kluczowymi polami.

users

Opis: Przechowuje dane zarejestrowanych użytkowników systemu, w tym ich dane osobowe (imię, nazwisko), unikalny nick, adres e-mail służący do logowania, bezpiecznie hashowane hasło oraz przypisaną rolę (np. określającą uprawnienia). Zawiera również znaczniki czasu utworzenia i ostatniej aktualizacji rekordu.

Kluczowe Pola: id, first_name, last_name, nick, email, password, role.

locales

Opis: Definiuje i przechowuje informacje o miejscach (lokalizacjach), w których mogą odbywać się wydarzenia. Zawiera nazwę miasta oraz unikalną nazwę konkretnego obiektu lub miejsca.

Kluczowe Pola: id, city, name.

categories

Opis: Przechowuje definicje kategorii, do których można przypisać wydarzenia w celu ich klasyfikacji (np. "Konferencja", "Warsztat"). Zawiera unikalną nazwę kategorii oraz jej unikalny opis.

Kluczowe Pola: id, name, description.

events

Opis: Centralna tabela przechowująca szczegółowe informacje o wydarzeniach. Zawiera unikalną nazwę i opis wydarzenia, jego cenę, dokładne daty i godziny rozpoczęcia oraz zakończenia. Posiada również powiązania (klucze obce) z tabelami locales (wskazując miejsce) i categories (wskazując kategorię).

Kluczowe Pola: id, locale_id, category_id, name, description, price, started_at, ended_at.

event_tickets

Opis: Tabela łącząca (junction table), która reprezentuje rejestrację (zakup biletu) użytkownika (users) na konkretne wydarzenie (events). Przechowuje cenę, po jakiej bilet został nabyty (może różnić się od bazowej ceny wydarzenia) oraz zapewnia unikalność rejestracji (jeden użytkownik może mieć tylko jeden bilet na dane wydarzenie).

Kluczowe Pola: id, event_id, user_id, price.

prelegents

Opis: Przechowuje informacje o prelegentach lub mówcach występujących na wydarzeniach. Zawiera unikalną nazwę prelegenta, jego opis (np. biografia, tematyka wystąpień) oraz powiązanie (klucz obcy) z kontem użytkownika w tabeli users (jeśli prelegent jest również użytkownikiem systemu).

Kluczowe Pola: id, user_id, name, description.

event_prelegents

Opis: Tabela łącząca, która przypisuje prelegentów (z tabeli prelegents) do konkretnych wydarzeń (z tabeli events), definiując, kto występuje na danym wydarzeniu. Zapewnia unikalność przypisania (jeden prelegent może być przypisany do danego wydarzenia tylko raz).

Kluczowe Pola: id, event_id, prelegent_id.

resources

Opis: Definiuje zasoby (np. sprzęt AV, projektory, sale konferencyjne, materiały), które mogą być wykorzystywane lub rezerwowane na potrzeby organizacji wydarzeń. Zawiera unikalną nazwę i opis zasobu.

Kluczowe Pola: id, name, description.

event_resources

Opis: Tabela łącząca, która przypisuje zasoby (z tabeli resources) do konkretnych wydarzeń (z tabeli events), wskazując, jakie zasoby są potrzebne, używane lub zarezerwowane dla danego wydarzenia. Gwarantuje unikalność powiązania.

Kluczowe Pola: id, event_id, resource_id.

sponsors

Opis: Przechowuje informacje o sponsorach wspierających wydarzenia. Zawiera unikalną nazwę sponsora oraz jego opis.

Kluczowe Pola: id, name, description.

event_sponsors

Opis: Tabela łącząca, która przypisuje sponsorów (z tabeli sponsors) do konkretnych wydarzeń (z tabeli events), wskazując, kto sponsoruje dane wydarzenie. Zapewnia unikalność powiązania.

Kluczowe Pola: id, event_id, sponsor_id.

caterings

Opis: Definiuje opcje lub dostawców cateringu (obsługi gastronomicznej) dostępnych dla wydarzeń. Zawiera unikalną nazwę opcji/dostawcy oraz jej/jego opis.

Kluczowe Pola: id, name, description.

event_caterings

Opis: Tabela łącząca, która przypisuje opcje cateringowe (z tabeli caterings) do konkretnych wydarzeń (z tabeli events), określając, jaka obsługa gastronomiczna jest przewidziana dla danego wydarzenia. Gwarantuje unikalność przypisania.

Kluczowe Pola: id, event_id, catering_id.

12. Role Użytkowników

System "Menedżer Wydarzeń" definiuje różne poziomy dostępu i uprawnień dla użytkowników poprzez przypisanie im jednej z poniższych ról. Role są reprezentowane w bazie danych (w tabeli users, w kolumnie role) za pomocą wartości liczbowych:

1. MEMBER (Członek/Uczestnik) - Wartość: 1

Opis: Jest to podstawowa rola dla standardowego, zarejestrowanego użytkownika aplikacji. Użytkownicy z tą rolą mogą zazwyczaj przeglądać szczegóły wydarzeń, zarządzać własnym profilem oraz potencjalnie rejestrować się na wydarzenia (kupować bilety).

2. PRELEGENT (Prelegent/Mówca) - Wartość: 2

Opis: Rola przypisywana użytkownikom, którzy są również prelegentami występującymi na wydarzeniach.

3. ADMINISTRATOR (Administrator) - Wartość: 3

Opis: Najwyższa rola w systemie, posiadająca pełne uprawnienia administracyjne. Administratorzy mogą zarządzać wszystkimi aspektami aplikacji, w tym: tworzeniem, edycją i usuwaniem *wszystkich* wydarzeń (nie tylko własnych), zarządzaniem kontami wszystkich użytkowników (edycja, usuwanie, zmiana ról), zarządzaniem kategoriami, lokalizacjami, zasobami, sponsorami, cateringiem oraz potencjalnie konfiguracją systemu. Mają dostęp do wszystkich danych i funkcji administracyjnych.

13. Struktura Kodu

Aplikacja "Menedżer Wydarzeń" jest podzielona na dwa główne komponenty: backend (API) i frontend (interfejs użytkownika), każdy z własną, logiczną strukturą kodu.

13.1. Struktura Kodu Backendu (Node.js / Express):

Backend aplikacji jest zorganizowany zgodnie ze wzorcami powszechnie stosowanymi w aplikacjach Express.js, często nawiązującymi do architektury Model-View-Controller

(MVC) lub jej wariantów (np. Model-Service-Controller), z wyraźnym podziałem odpowiedzialności:

server.js (lub index.js): Plik wejściowy

Odpowiedzialność: Inicjalizacja serwera Express, ładowanie konfiguracji (np. z .env), konfiguracja globalnych middleware (np. cors, parser ciała żądania `express.json()`, `express.urlencoded()`), konfiguracja połączenia z bazą danych (Knex), ładowanie i montowanie głównych routerów, uruchomienie serwera nasłuchującego na określonym porcie.

/config (Katalog): Konfiguracja

Odpowiedzialność: Przechowywanie plików konfiguracyjnych, np. konfiguracja połączenia Knex z bazą danych (`knexfile.js`), ustawienia aplikacji, klucze (np. `JWT_SECRET` ładowany z .env).

/routes (Katalog): Definicje Tras (Routing)

Odpowiedzialność: Definiowanie endpointów API. Zazwyczaj zawiera oddzielne pliki dla każdego zasobu/modułu (np. `event.routes.js`, `auth.routes.js`, `user.routes.js`). Pliki te używają `express.Router()` do grupowania tras i przypisują je do odpowiednich funkcji w kontrolerach, stosując po drodze niezbędne middleware (np. walidacyjne, autoryzacyjne).

/controllers (Katalog): Kontrolery

Odpowiedzialność: Obsługa logiki związanej bezpośrednio z żądaniem HTTP. Odbierają żądanie, wywołują odpowiednie funkcje w serwisach (przekazując przetworzone dane wejściowe, np. `req.body`, `req.params`), a następnie formatują i wysyłają odpowiedź HTTP (sukces lub błąd) do klienta. Każdy zasób ma zwykle swój kontroler (np. `event.controller.js`).

/services (Katalog): Serwisy (Logika Biznesowa)

Odpowiedzialność: Hermetyzacja głównej logiki biznesowej aplikacji, niezależnej od protokołu HTTP. Serwisy koordynują operacje, wykonują złożone zadania, wchodzi w interakcje z modelami (lub bezpośrednio z bazą danych przez Knex) w celu pobrania/zapisu danych. Przykłady: `event.service.js`, `auth.service.js`. W prostszych aplikacjach logika ta może znajdować się bezpośrednio w kontrolerach.

/models (Katalog) lub Logika DB w Serwisach: Modele / Dostęp do Danych

Odpowiedzialność: Reprezentacja struktury danych i interakcja z bazą danych. W kontekście Knex.js, może to oznaczać funkcje lub klasy opakowujące zapytania Knex dla konkretnych tabel (`users`, `events` itp.), realizujące operacje CRUD na poziomie bazy. Czasem ta warstwa jest połączona z serwisami.

/middleware (Katalog): Middleware

Odpowiedzialność: Zawiera funkcje pośredniczące, które są wykonywane przed logiką kontrolera. Przykłady:

auth.middleware.js: Weryfikacja tokenu JWT i dołączanie danych użytkownika do obiektu req.

validation.middleware.js (lub definicje w /validators): Użycie express-validator do definiowania i uruchamiania reguł walidacji danych wejściowych. Błędy walidacji są tutaj przechwytywane i zwracane przed dotarciem do kontrolera.

error.middleware.js: Globalna obsługa błędów.

/validators (Katalog) (Opcjonalnie): Walidatory

Odpowiedzialność: Czasem reguły walidacyjne express-validator są wyodrębniane do osobnych plików w tym katalogu (np. event.validator.js), aby utrzymać czystość w plikach tras. Następnie są importowane i używane jako middleware w /routes.

/db (Katalog): Baza Danych

Odpowiedzialność: Zwykle zawiera podkatalogi zarządzane przez Knex:

/migrations: Pliki definiujące zmiany w schemacie bazy danych (tworzenie/modyfikacja tabel).

/seeds: Pliki do wypełniania bazy danych początkowymi danymi (np. role, kategorie, użytkownik admin).

/tests (Katalog): Testy

Odpowiedzialność: Zawiera pliki testów jednostkowych i integracyjnych (np. *.test.js), wykorzystujące Jest i Supertest do weryfikacji poprawności działania poszczególnych komponentów i całego API.

13.2. Struktura Kodu Frontendu (HTML/CSS/JS/Bootstrap):

Frontend, jako aplikacja jednostronicowa (SPA) bez ciężkiego frameworka, opiera się na modułowej strukturze JavaScript i dynamicznym renderowaniu treści:

index.html: Plik wejściowy HTML

Odpowiedzialność: Główny szkielet strony. Zawiera podstawową strukturę HTML (<head>, <body>), linki do CSS (Bootstrap, custom style.css), ładowanie głównego skryptu JS (js/main.js jako moduł) oraz kontenery (#navbar-container, #app-content, #footer-container) do dynamicznego wstrzykiwania treści.

/css (Katalog): Style CSS

Odpowiedzialność: Zawiera niestandardowe style (style.css), które rozszerzają lub modyfikują domyślne style Bootstrap, dostosowując wygląd aplikacji.

/js (Katalog): Skrypty JavaScript

Odpowiedzialność: Cała logika aplikacji po stronie klienta. Struktura wewnątrz tego katalogu może wyglądać następująco:

main.js: Główny skrypt aplikacji

Punkt startowy. Inicjalizuje router, renderuje statyczne komponenty (np. nawigację, stopkę), potencjalnie ustawia globalne event listenery.

router.js: Router po stronie klienta

Odpowiada za obsługę zmian w URL (np. hash lub History API), mapowanie ścieżek na odpowiednie moduły widoków i wywoływanie ich funkcji renderujących.

/views (Podkatalog): Moduły Widoków

Każdy plik (np. loginView.js, eventListView.js, eventFormView.js) odpowiada za renderowanie konkretnej "strony" lub sekcji aplikacji w kontenerze #app-content. Moduł taki zazwyczaj:

- Definiuje funkcję render().

- Pobiera potrzebne dane z API (korzystając z modułu serwisu API).

- Generuje HTML dla widoku (np. za pomocą template strings).

- Dołącza event listenery do elementów w swoim widoku (np. obsługa formularzy, kliknięcia przycisków).

- Wstawia wygenerowany HTML do #app-content.

helpers.js: Narzędzia/Pomocnicze

Zawiera pomocnicze, reużywalne funkcje używane w różnych częściach aplikacji (np. formatowanie dat, proste funkcje DOM).

/node_modules (Katalog): Zależności

Zawiera zainstalowane biblioteki (Bootstrap, Bootstrap Icons) zarządzane przez npm. Pliki z tego katalogu są linkowane w index.html.