**Główne cele projektu systemu zarządzania projektami studenckimi**

**Cel i opis systemu**

System ma na celu stworzenie centralnej platformy umożliwiającej studentom publikowanie i zarządzanie swoimi projektami, a nauczycielom ich weryfikację i akceptację. Projekty będą uporządkowane według kategorii tematycznych, a każdy użytkownik z odpowiednimi uprawnieniami będzie miał dostęp do ich edycji lub oceny.

System zapewni intuicyjny interfejs użytkownika, który umożliwi łatwe wyszukiwanie i przeglądanie projektów bez konieczności ich pobierania na komputer. Logowanie do systemu będzie realizowane wyłącznie przez MS Teams i Active Directory uczelni, co zapewni bezpieczeństwo i kontrolę nad dostępem. Publikacja projektów wymaga zatwierdzenia przez nauczycieli, co zapewni wysoką jakość udostępnianych materiałów.

**Uzasadnienie biznesowe**

Centralizacja zarządzania projektami studenckimi w jednym systemie znacząco usprawni proces nauczania i oceniania prac studentów. Integracja z istniejącymi systemami uczelnianymi (MS Teams, Active Directory) eliminuje potrzebę tworzenia nowych kont i haseł, a proces weryfikacji przez nauczycieli zapewnia kontrolę jakości materiałów.

**Główne funkcjonalności**

1. **Zarządzanie użytkownikami i uprawnieniami**
   * System ról: Administrator, Nauczyciel, Student, Gość
   * Logowanie przez MS Teams i Active Directory
2. **Zarządzanie projektami**
   * Tworzenie, edycja i publikacja projektów
   * Weryfikacja i zatwierdzanie projektów
   * Kategoryzacja i tagowanie
   * Przeglądanie bez konieczności pobierania
3. **Wyszukiwanie i filtrowanie**
   * Zaawansowane opcje wyszukiwania
   * Filtrowanie według kategorii, tagów, autorów, statusu
4. **Administracja systemem**
   * Zarządzanie kategoriami i tagami
   * Moderacja treści
   * Generowanie raportów

**Oczekiwane korzyści**

* Ujednolicony system przechowywania i prezentowania projektów
* Łatwy dostęp do materiałów dydaktycznych
* Przejrzysty proces weryfikacji i zatwierdzania projektów
* Oszczędność czasu nauczycieli i studentów
* Kontrola jakości publikowanych materiałów

**Specyfikacja techniczna projektu**

**1. Technologie**

**Frontend**

* **Framework**: Vue.js 3 (Composition API)
* **CSS Framework**: Tailwind CSS
* **Zarządzanie stanem**: Pinia/Vuex
* **Routing**: Vue Router

**Uzasadnienie**: Vue.js zapewnia lekkość, elastyczność oraz łatwą krzywą uczenia się. Composition API wprowadzone w Vue 3 umożliwia tworzenie bardziej modułowego i czytelnego kodu. Tailwind CSS pozwala na szybkie tworzenie spójnego i responsywnego interfejsu poprzez komponowanie klas CSS.

**Backend**

* **Framework**: Node.js + NestJS
* **ORM**: Prisma
* **API**: RESTful

**Uzasadnienie**: NestJS dostarcza solidną strukturę do budowy skalowalnych aplikacji serwerowych w TypeScript. Wspiera modułową architekturę i dependency injection, co ułatwia organizację kodu. Prisma ORM zapewnia type-safe zapytania i ułatwia migracje bazy danych.

**Baza danych**

* **System**: PostgreSQL

**Uzasadnienie**: PostgreSQL to solidna, relacyjna baza danych o otwartym kodzie, która doskonale sprawdza się w aplikacjach z kompleksowym modelem danych, oferując zaawansowane możliwości zapytań i integralność danych.

**Autoryzacja**

* **Logowanie wyłącznie przez MS Teams i Active Directory uczelni**
* **Wielopoziomowy system dostępu** (Administrator, Nauczyciel, Student, Gość)

**Uzasadnienie**: Integracja z MS Teams i Active Directory uczelni eliminuje potrzebę tworzenia osobnego systemu rejestracji i logowania. Użytkownicy mogą korzystać z tych samych danych uwierzytelniających, których używają na co dzień, co upraszcza dostęp do aplikacji i zwiększa bezpieczeństwo.

**Infrastruktura**

* **Hostowanie**: Konfiguracja umożliwiająca zarówno lokalny development jak i wdrożenie na docelowy serwer uczelniany
* **Konteneryzacja**: Docker
* **Zabezpieczenia**: HTTPS, certyfikat SSL
* **Dostępność**: Przez adres WWW

**Uzasadnienie**: Konteneryzacja z Docker zapewnia jednolite środowisko dla rozwoju, testowania i produkcji. HTTPS i certyfikat SSL zapewniają bezpieczną komunikację i zwiększają zaufanie użytkowników do systemu.

**2. Architektura systemu**

**Schemat architektury**

System jest zbudowany w architekturze trójwarstwowej:

1. **Warstwa prezentacji** (frontend) - aplikacja SPA w Vue.js
2. **Warstwa logiki biznesowej** (backend) - API NestJS
3. **Warstwa danych** (baza danych) - PostgreSQL

**Uzasadnienie**: Podział na trzy warstwy pozwala na niezależny rozwój każdej z nich, zwiększa modularność i ułatwia testowanie. Jest to sprawdzony standard w nowoczesnych aplikacjach webowych, który umożliwia skalowanie poszczególnych komponentów w zależności od potrzeb.

**Komunikacja między warstwami**

* Frontend komunikuje się z backendem poprzez RESTful API
* Backend komunikuje się z bazą danych poprzez Prisma ORM
* Autoryzacja wykorzystuje MS Teams i Active Directory

**Uzasadnienie**: RESTful API zapewnia standaryzowany sposób komunikacji między frontendem a backendem, niezależny od technologii. Prisma ORM znacząco upraszcza operacje na bazie danych i zapewnia typebezpieczne zapytania.

**3. Bezpieczeństwo**

**Autoryzacja i uwierzytelnianie**

* Integracja z MS Teams i Active Directory uczelni
* Tokeny JWT do autoryzacji API
* Role i uprawnienia oparte na RBAC

**Uzasadnienie**: Wykorzystanie istniejących mechanizmów uwierzytelniania uczelni zwiększa bezpieczeństwo i upraszcza proces logowania. Tokeny JWT zapewniają bezstatusową autoryzację API, a RBAC precyzyjnie kontroluje dostęp do funkcji systemu.

**Zabezpieczenia na poziomie infrastruktury**

* HTTPS z certyfikatem SSL
* Bezpieczne zarządzanie sekretami
* Konteneryzacja z Docker

**Uzasadnienie**: HTTPS i SSL zapewniają szyfrowanie komunikacji między klientem a serwerem. Konteneryzacja izoluje poszczególne komponenty systemu, a bezpieczne zarządzanie sekretami chroni wrażliwe dane konfiguracyjne.

**Ochrona danych**

* Walidacja danych wejściowych
* Ochrona przed atakami CSRF, XSS, SQL Injection
* Logowanie zdarzeń i monitorowanie

**Uzasadnienie**: Kompleksowa ochrona danych obejmuje walidację wszystkich danych wejściowych, zabezpieczenia przed popularnymi atakami webowymi oraz monitorowanie systemu w celu szybkiego wykrywania potencjalnych naruszeń bezpieczeństwa.

**Struktura projektu**

**Struktura katalogów**

Copy

projekt-studencki/

│

├── frontend/ # Aplikacja Vue.js

│ ├── public/ # Statyczne pliki

│ │ ├── favicon.ico

│ │ └── index.html

│ │

│ ├── src/ # Kod źródłowy frontendu

│ │ ├── assets/ # Zasoby statyczne (obrazy, fonty)

│ │ ├── components/ # Komponenty Vue

│ │ │ ├── common/ # Wspólne komponenty UI

│ │ │ ├── projects/ # Komponenty związane z projektami

│ │ │ ├── users/ # Komponenty związane z użytkownikami

│ │ │ └── admin/ # Komponenty panelu administracyjnego

│ │ │

│ │ ├── views/ # Widoki stron

│ │ ├── router/ # Konfiguracja routingu

│ │ ├── store/ # Store Pinia

│ │ ├── services/ # Serwisy komunikacji z API

│ │ ├── utils/ # Funkcje pomocnicze

│ │ ├── App.vue # Główny komponent

│ │ └── main.js # Punkt wejścia

│ │

│ ├── .env # Zmienne środowiskowe

│ ├── package.json # Zależności npm

│ ├── vite.config.js # Konfiguracja Vite

│ └── tailwind.config.js # Konfiguracja Tailwind CSS

│

├── backend/ # API NestJS

│ ├── src/ # Kod źródłowy backendu

│ │ ├── main.ts # Punkt wejścia

│ │ ├── app.module.ts # Główny moduł aplikacji

│ │ │

│ │ ├── modules/ # Moduły funkcjonalne

│ │ │ ├── users/ # Moduł użytkowników

│ │ │ ├── projects/ # Moduł projektów

│ │ │ ├── categories/ # Moduł kategorii

│ │ │ └── reports/ # Moduł raportów

│ │ │

│ │ ├── common/ # Wspólne elementy

│ │ │ ├── filters/ # Filtry wyjątków

│ │ │ ├── guards/ # Guardy autoryzacji

│ │ │ ├── interceptors/ # Interceptory

│ │ │ └── decorators/ # Dekoratory niestandardowe

│ │ │

│ │ └── config/ # Konfiguracje

│ │

│ ├── prisma/ # Konfiguracja Prisma ORM

│ │ ├── schema.prisma # Schema bazy danych

│ │ └── migrations/ # Migracje bazy danych

│ │

│ ├── .env # Zmienne środowiskowe

│ └── package.json # Zależności npm

│

├── docker/ # Konfiguracja Docker

│ ├── docker-compose.yml # Główny plik compose

│ ├── docker-compose.dev.yml # Konfiguracja dla środowiska dev

│ ├── docker-compose.prod.yml # Konfiguracja dla produkcji

│ │

│ ├── frontend/ # Konfiguracja dla frontendu

│ │ ├── Dockerfile

│ │ └── nginx.conf

│ │

│ ├── backend/ # Konfiguracja dla backendu

│ │ └── Dockerfile

│ │

│ ├── database/ # Konfiguracja dla bazy danych

│ │ └── init.sql

│ │

│ └── nginx/ # Konfiguracja dla reverse proxy

│ ├── Dockerfile

│ ├── nginx.conf

│ └── ssl/

│

└── docs/ # Dokumentacja

├── api/ # Dokumentacja API

├── user-guide/ # Instrukcja użytkownika

├── dev-guide/ # Instrukcja dla developera

└── deployment/ # Instrukcja wdrożenia

**Moduły systemu**

**1. Moduł Użytkowników**

**Role użytkowników**

* **Administrator**: Pełny dostęp do systemu, zarządzanie użytkownikami, generowanie raportów
* **Nauczyciel**: Weryfikacja i zatwierdzanie/odrzucanie projektów, zapewnienie kontroli jakości
* **Student**: Tworzenie i edycja własnych projektów przed zatwierdzeniem
* **Gość**: Tylko przeglądanie zatwierdzonych projektów

**Uzasadnienie**: System ról i uprawnień jest kluczowy dla kontroli dostępu do różnych funkcjonalności aplikacji. Podział na administratorów, nauczycieli, studentów i gości odpowiada rzeczywistym rolom w środowisku akademickim i zapewnia odpowiedni poziom kontroli nad treściami.

**Funkcjonalność**

* Logowanie wyłącznie przez MS Teams / Active Directory uczelni (bez rejestracji)
* Zarządzanie profilami użytkowników
* Kontrola dostępu oparta na rolach (RBAC)
* Bezpieczeństwo i kontrola nad dostępem do systemu

**2. Moduł Projektów**

**Funkcjonalność**

* Tworzenie nowych projektów (student)
* Edycja projektów przed zatwierdzeniem (student)
* Weryfikacja i akceptacja/odrzucenie projektów (nauczyciel)
* Przeglądanie projektów z możliwością filtrowania (wszyscy)
* Szczegółowy widok projektu z pełnym opisem i metadanymi
* Przeglądanie projektów bez konieczności ich pobierania
* Automatyczne kategoryzowanie projektów według tematyki

**Uzasadnienie**: Centralna funkcjonalność systemu, umożliwiająca zarządzanie całym cyklem życia projektu - od utworzenia, przez recenzję, aż po publikację. Możliwość przeglądania projektów bez pobierania oszczędza miejsce na dysku użytkowników i zwiększa dostępność materiałów.

**Struktura projektu**

* Tytuł
* Opis
* Kategoria
* Data utworzenia
* Autor
* Status (nowy, w weryfikacji, zatwierdzony, odrzucony)
* Pliki/załączniki
* Komentarze od nauczyciela
* Tagi

**3. Moduł Administracyjny**

**Funkcjonalność**

* Zarządzanie użytkownikami (dodawanie, edycja, usuwanie)
* Zarządzanie kategoriami projektów
* Moderacja treści
* Generowanie raportów (popularność projektów, aktywność użytkowników)

**Uzasadnienie**: Moduł administracyjny zapewnia narzędzia do zarządzania całym systemem, pozwalając administratorom na kontrolę użytkowników, kategorii i treści. Generowanie raportów umożliwia analizę wykorzystania systemu i podejmowanie decyzji na podstawie danych, np. identyfikację najbardziej aktywnych studentów lub najpopularniejszych kategorii projektów.

**Model danych**

**Główne encje**

1. **User** (Użytkownik)
   * id, email, firstName, lastName, role, adUserId, msTeamsId, createdAt, updatedAt
2. **Project** (Projekt)
   * id, title, description, status, createdAt, updatedAt, publishedAt, authorId, approverId, categoryId
3. **Category** (Kategoria)
   * id, name, description, parentId, createdAt, updatedAt
4. **File** (Plik)
   * id, filename, originalName, mimeType, size, path, createdAt, projectId
5. **Comment** (Komentarz)
   * id, content, createdAt, updatedAt, authorId, projectId
6. **Tag** (Tag)
   * id, name, createdAt
7. **ProjectTag** (Relacja Projekt-Tag)
   * projectId, tagId
8. **Report** (Raport)
   * id, name, type, parameters, result, createdAt

**Uzasadnienie**: Rozbudowany model danych z wieloma powiązanymi encjami umożliwia przechowywanie wszystkich niezbędnych informacji o projektach, użytkownikach i ich interakcjach. Hierarchiczna struktura kategorii pozwala na logiczną organizację projektów, a model tagów zapewnia elastyczne grupowanie.