WYŻSZA SZKOŁA
INFORMATYKI i ZARZĄDZANIA
z siedzibą w Rzeszowie

KOLEGIUM INFORMATYKI STOSOWANEJ

Kierunek: PROGRAMOWANIE

Grupa: L1

Hlib Hospodarysko 69976, Tsimafei Andropau 70589 i Uladzislau Sopat 71048

Generator konspektów

Prowadzący: inż. Paweł Janda

Praca projektowa. Programowanie Full-Stack

Rzeszów 2026

Wstęp

Proces tworzenia konspektów oraz materiałów do nauki jest często czasochłonny i wymaga dobrej organizacji informacji. Uczniowie, studenci oraz nauczyciele regularnie przygotowują opracowania na różne tematy, korzystając z wielu źródeł – podręczników, artykułów naukowych, stron internetowych czy notatek własnych. W natłoku treści łatwo się pogubić, a samo uporządkowanie najważniejszych informacji w logiczny i przejrzysty sposób bywa trudne i wymaga dodatkowego czasu.

W praktyce wiele osób tworzy konspekty w sposób niesystematyczny, kopiując fragmenty tekstów lub sporządzając chaotyczne notatki, które nie zawsze odzwierciedlają strukturę zagadnienia. Brak jasnego planu może prowadzić do pominięcia istotnych kwestii, nieczytelnej struktury pracy oraz trudności w późniejszym przyswajaniu materiału.

Dlatego potrzebne jest narzędzie, które usprawni i uporządkuje proces przygotowywania konspektów. Generator konspektów umożliwia szybkie tworzenie logicznie zorganizowanego planu na wybrany temat, pomagając użytkownikowi skupić się na najważniejszych zagadnieniach. Dzięki takiemu rozwiązaniu przygotowanie materiałów staje się bardziej efektywne, przejrzyste i mniej stresujące, a użytkownik zyskuje większą kontrolę nad procesem nauki lub opracowywania treści.

Kod źródłowy: https://github.com/WladosSopot/Web-Project/tree/main

Opis założeń projektu

Cel projektu

Celem projektu jest stworzenie systemu, który ułatwi tworzenie uporządkowanych konspektów na podstawie tematu wprowadzonego przez użytkownika. System ma wspierać sprawne i przejrzyste przygotowywanie planów wypowiedzi, referatów oraz materiałów edukacyjnych.

Takie narzędzie może znacząco usprawnić proces opracowywania treści – od momentu podania tematu, przez wygenerowanie logicznie uporządkowanej struktury, aż po możliwość ponownego przeglądania wcześniej utworzonych konspektów. Dodatkowo system pozwoli użytkownikowi oszczędzić czas oraz uniknąć chaosu informacyjnego podczas organizowania materiału.

Proces tworzenia konspektu wymaga logicznego myślenia i odpowiedniego uporządkowania zagadnień. Brak jasnej struktury często prowadzi do pominięcia istotnych elementów tematu lub nieczytelnego układu treści. Dlatego potrzebne jest rozwiązanie, które umożliwi szybkie i uporządkowane generowanie planu wypowiedzi.

Aby zrealizować ten cel, system powinien zawierać następujące elementy:

1. Mechanizm generowania konspektu na podstawie tematu wprowadzonego przez użytkownika.
2. Moduł zapisywania historii wygenerowanych konspektów.
3. Prosty i intuicyjny interfejs umożliwiający wprowadzanie tematu oraz przeglądanie wyników.
4. Mechanizmy przetwarzania danych pozwalające na zapisywanie i odczytywanie historii.
5. Spójną komunikację pomiędzy warstwą interfejsu użytkownika a mechanizmem przechowywania danych.

Wymagania funkcjonalne i niefunkcjonalne

Wymagania funkcjonalne

1. **Wprowadzanie tematu.** System powinien umożliwiać użytkownikowi wprowadzenie tematu konspektu w polu tekstowym.
2. **Generowanie konspektu.** System powinien automatycznie generować uporządkowany konspekt zawierający wprowadzenie, rozwinięcie oraz podsumowanie.
3. **Wyświetlanie wyników.** System powinien prezentować wygenerowany konspekt w czytelnej i przejrzystej formie.
4. **Zapisywanie historii.** System powinien zapisywać informacje o wygenerowanych konspektach, w tym temat, treść oraz datę utworzenia.
5. **Przeglądanie historii.** System powinien umożliwiać użytkownikowi przeglądanie wcześniej wygenerowanych konspektów.
6. **Walidacja danych wejściowych.** System powinien sprawdzać poprawność wprowadzonego tematu (np. minimalna liczba znaków).
7. **Nawigacja w aplikacji.** System powinien umożliwiać przechodzenie pomiędzy stroną główną a stroną historii.

Wymagania niefunkcjonalne

1. **Spójność danych.**Każdy zapisany konspekt powinien posiadać jednolitą strukturę obejmującą temat, treść oraz datę utworzenia.
2. **Wydajność.**Generowanie konspektu oraz odczyt historii powinny odbywać się w czasie nieodczuwalnym dla użytkownika.
3. **Użyteczność.**Interfejs użytkownika powinien być prosty, intuicyjny i czytelny, umożliwiający korzystanie z aplikacji bez dodatkowych instrukcji.
4. **Niezawodność.**Dane zapisane w historii nie powinny być tracone podczas odświeżenia strony przeglądarki.
5. **Dostępność.**System powinien działać w nowoczesnych przeglądarkach internetowych bez konieczności instalowania dodatkowego oprogramowania.
6. **Możliwość rozbudowy.**Architektura systemu powinna umożliwiać przyszłą rozbudowę o dodatkowe funkcjonalności, takie jak logowanie użytkownika i zapis do bazy danych.

Opis struktury projektu

Struktura oraz opis techniczny

System składa się z trzech głównych warstw: warstwy prezentacji (frontend), warstwy logiki aplikacji oraz warstwy przechowywania danych.

- **Warstwa przechowywania danych** odpowiada za zapisywanie oraz odczytywanie informacji dotyczących wygenerowanych konspektów. W obecnej wersji projektu dane przechowywane są w relacyjnej bazie danych SQLite. Przechowywane informacje obejmują: temat konspektu, wygenerowaną treść oraz datę utworzenia wpisu. Struktura danych jest jednolita i zapisywana w postaci tabeli bazy danych. W przyszłości warstwa ta może zostać rozszerzona o dodatkowe mechanizmy, takie jak obsługa wielu użytkowników oraz ich autoryzacja.

- **Warstwa logiki aplikacji** odpowiada za przetwarzanie danych, generowanie konspektów oraz komunikację pomiędzy interfejsem użytkownika a mechanizmem przechowywania danych. Do jej głównych zadań należą: walidacja danych wejściowych (np. sprawdzanie minimalnej długości tematu), generowanie struktury konspektu na podstawie wprowadzonego tematu, zapisywanie wygenerowanych danych w pamięci przeglądarki, pobieranie i wyświetlanie historii konspektów. Takie rozdzielenie zwiększa czytelność kodu oraz umożliwia łatwą rozbudowę systemu.

- **Warstwa prezentacji (Interfejs użytkownika)** została zaimplementowana jako aplikacja webowa w technologii React. Jej zadaniem jest: umożliwienie użytkownikowi wprowadzenia tematu konspektu, uruchomienie procesu generowania, wyświetlenie wygenerowanego konspektu, zapewnienie dostępu do historii zapisanych konspektów, umożliwienie nawigacji pomiędzy stroną główną a stroną historii. Interfejs użytkownika został zaprojektowany w sposób prosty i intuicyjny. Aplikacja działa w przeglądarce internetowej i nie wymaga instalowania dodatkowego oprogramowania.

Baza danych

Baza danych systemu Generator konspektów składa się z dwóch głównych tabel: **Users** oraz **History**. Struktura ta umożliwia przechowywanie informacji o użytkownikach aplikacji oraz o wygenerowanych przez nich konspektach.

Tabela **Users** zawiera podstawowe dane identyfikacyjne użytkownika, takie jak identyfikator (ID), nazwa użytkownika, adres e-mail oraz hasło (przechowywane w postaci zaszyfrowanej). Każdy użytkownik posiada unikalny identyfikator, który pozwala na powiązanie go z wygenerowanymi konspektami.

Tabela **History** przechowuje informacje o wygenerowanych konspektach, w tym temat, treść konspektu, datę utworzenia oraz identyfikator użytkownika. Relacja pomiędzy tabelami realizowana jest za pomocą klucza obcego (*UserId*), który łączy wpisy w tabeli **History** z odpowiednim użytkownikiem w tabeli **Users**. Zapewnia to spójność danych oraz umożliwia przechowywanie indywidualnej historii dla każdego użytkownika.

Warstwa logiczna

Podwarstwa `Domain`

Odpowiada za reprezentację obiektów z bazy danych w postaci modeli domenowych. Zawiera klasy encji odzwierciedlające strukturę tabel w bazie danych oraz definiuje ich właściwości i relacje. Warstwa ta stanowi centralny punkt logiki biznesowej aplikacji.

Podwarstwa `API`

Odpowiada za implementację interfejsu REST API. Zawiera kontrolery obsługujące żądania HTTP (GET, POST, PUT, DELETE) oraz mapowanie danych pomiędzy warstwą prezentacji a warstwą domenową. Umożliwia komunikację klienta z systemem.

Podwarstwa `Infrastructure`

Odpowiada za implementację mechanizmu komunikacji z zewnętrznym API **Gemini**. Zawiera klasy odpowiedzialne za wysyłanie zapytań do modelu, odbieranie odpowiedzi oraz ich przetwarzanie na potrzeby systemu. Warstwa ta izoluje logikę integracji z usługą zewnętrzną od pozostałych części aplikacji.

Opis stosu technologicznego

Języki

- System został stworzony w języku **TypeScript** z użyciem frameworku **NestJS**
- Interfejs użytkownika został stworzony z wykorzystaniem **HTML/CSS** z użyciem frameworku **React**

Bazy danych

- Baza danych została stworzona przy użyciu języka **SQLite**

Narzędzia

- Microsoft Visual Studio Code
- Postman

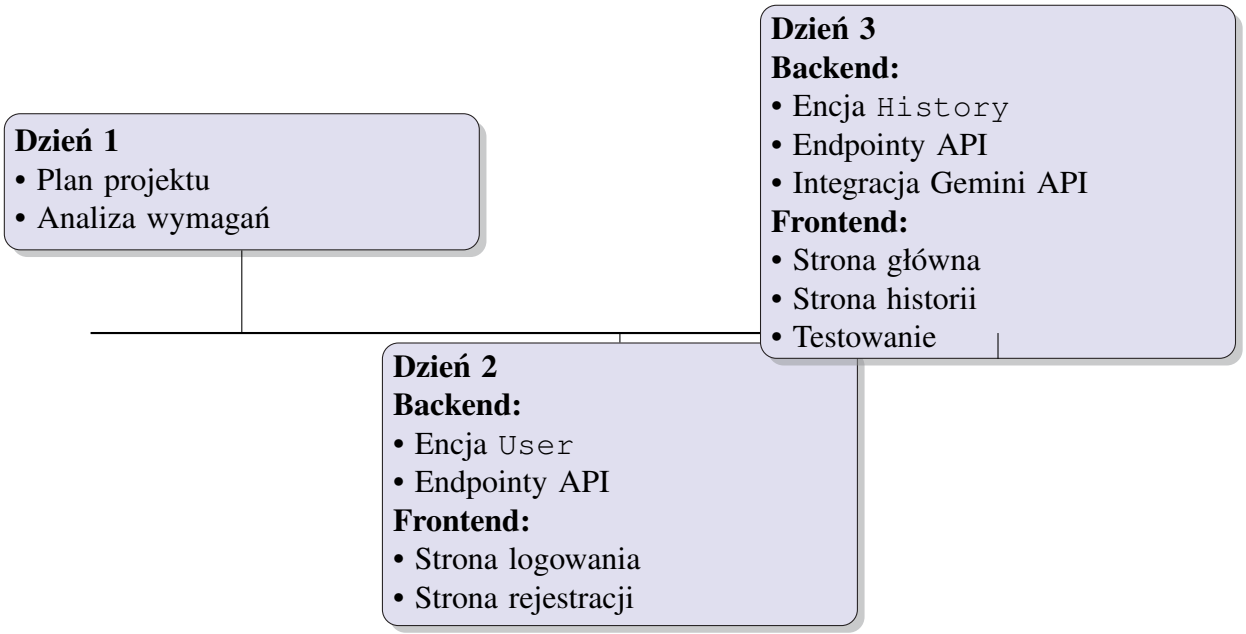
Opis endpointów API

- **POST /auth/register**
  Endpoint odpowiedzialny za rejestrację nowego użytkownika. Przyjmuje dane rejestracyjne (np. e-mail oraz hasło), waliduje je i tworzy nowe konto w bazie danych.
- **POST /auth/login**
  Endpoint odpowiedzialny za uwierzytelnianie użytkownika. Weryfikuje poprawność danych logowania i zwraca token autoryzacyjny umożliwiający dostęp do chronionych zasobów systemu.
- **GET /history**
  Endpoint służący do pobierania historii wygenerowanych konspektów zalogowanego użytkownika. Zwraca listę zapisanych wpisów wraz z tematem, treścią oraz datą utworzenia.

- **DELETE /history/{id}**
  Endpoint umożliwiający usunięcie wybranego wpisu z historii na podstawie jego identyfikatora.
- **POST /ai/request**
  Endpoint odpowiedzialny za wysłanie zapytania do API Gemini w celu wygenerowania konspektu na podstawie podanego tematu. Zwraca wygenerowaną treść oraz zapisuje ją w historii użytkownika.

Process powstawania projektu

Harmonogram



Planowanie i estymacja zadań

Proces planowania projektu został przeprowadzony w sposób iteracyjny, z podziałem prac na trzy główne etapy realizacyjne. Zakres funkcjonalny obejmował implementację systemu generowania konspektów, obsługę użytkowników, zapis historii oraz integrację z zewnętrznym API.

Identyfikacja zadań

Na podstawie analizy wymagań wyodrębniono następujące obszary prac:

- zaprojektowanie architektury aplikacji,
- implementacja encji `User` oraz `History`,
- przygotowanie endpointów REST API,
- integracja z API Gemini,
- implementacja warstwy frontendowej,
- testowanie i poprawa błędów.

Estymacja czasowa

Estymacja została wykonana metodą ekspercką na podstawie przewidywanej złożoności systemu.

Dzień 1

- analiza wymagań,
- zaprojektowanie struktury projektu,
- przygotowanie modeli danych,
- konfiguracja środowiska.

Dzień 2

- implementacja encji `User`,
- utworzenie endpointów rejestracji i logowania.
- implementacja strony logowania,
- implementacja strony rejestracji,
- integracja formularzy z API.

Dzień 3

- implementacja encji `History`,
- utworzenie endpointów zapisu i odczytu historii,
- integracja z API Gemini.
- implementacja strony głównej,
- implementacja strony historii,
- testowanie integracyjne systemu.

Podział zadań wśród zespołu

- **Tsimafei Andropau (70589)** – odpowiedzialny za implementację większości warstwy frontendowej aplikacji oraz przygotowanie dokumentacji projektowej.
- **Uladzislau Sopat (71048)** – odpowiedzialny za integrację z API Gemini, przeprowadzenie testów systemu oraz koordynację prac zespołu (rola lidera zespołu). Uczestniczył również w wybranych zadaniach związanych z frontendem i backendem.
- **Hlib Hospodarysko (69976)** – odpowiedzialny za implementację większości warstwy backendowej, zaprojektowanie struktury bazy danych oraz analizę wymagań systemowych.

Opis przypadków testowych

Przypadek testowy 1: Rejestracja nowego użytkownika

```
Funkcjonalność: Rejestracja użytkownika w systemie

  Jako nowy użytkownik
  Chcę mieć możliwość utworzenia konta
  Aby móc generować i zapisywać konspekty

  Kontekst:
    Zakładając, że znajduję się na stronie „Rejestracja”

  Scenariusz: Pomyślna rejestracja
    Gdy wpiszę poprawny username w polu „Username”
    Oraz wpiszę hasło spełniające wymagania bezpieczeństwa
    I kliknę przycisk „Zarejestruj”
    Wtedy powinienem zobaczyć komunikat o pomyślnej rejestracji
    Oraz powinienem zostać przekierowany na stronę logowania
```

Przypadek testowy 2: Generowanie i zapis konspektu

```
Funkcjonalność: Generowanie konspektu z wykorzystaniem API Gemini

  Jako zalogowany użytkownik
  Chcę wygenerować konspekt na wybrany temat
  Aby móc zapisać go w historii i wykorzystać w przyszłości

  Kontekst:
    Zakładając, że jestem zalogowany
    Oraz znajduję się na stronie głównej aplikacji

  Scenariusz: Pomyślne wygenerowanie konspektu
    Gdy wpiszę temat „Architektura systemów informatycznych” w polu „Temat”
    I kliknę przycisk „Generuj”
    Wtedy system powinien wysłać zapytanie do API Gemini
    Oraz powinienem zobaczyć wygenerowaną treść konspektu
    Oraz konspekt powinien zostać zapisany w historii użytkownika
```

Podsumowanie

W wyniku realizacji projektu zostało stworzone proste i funkcjonalne narzędzie umożliwiające generowanie oraz przechowywanie konspektów na podstawie tematów wprowadzanych przez użytkownika. System pozwala na szybkie tworzenie uporządkowanej struktury wypowiedzi, co usprawnia proces przygotowywania materiałów edukacyjnych.

Zaimplementowana aplikacja umożliwia generowanie konspektów, zapisywanie ich w historii oraz ponowne przeglądanie wcześniej utworzonych treści. Dzięki zastosowaniu przejrzystego interfejsu użytkownika system jest intuicyjny i łatwy w obsłudze. Warstwowa struktura projektu zapewnia czytelność kodu oraz możliwość dalszej rozbudowy.

Bibliografia

[1] https://stackoverflow.com/

[2] https://www.w3schools.com/

[3] https://learn.microsoft.com/

[4] https://www.codecademy.com/

[5] https://www.reddit.com/

[6] https://ai.google.dev/gemini-api/docs/quickstart/