



AGH

AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE
IEiT

INSTYTUT INFORMATYKI

Projekt dyplomowy

Uniwersalny kategoryzator danych
Universal data categorizer

Autor:	<i>Tomasz Marek</i>
Kierunek studiów:	Informatyka niestacjonarne
Opiekun pracy:	<i>dr inż. Włodzimierz Funika</i>

Kraków, 2022

Spis Treści

• Wstęp	3
○ Cel pracy	3
○ Charakterystyka problemu	3
○ Istniejące rozwiązania	4
• Projektowanie	6
○ Wstęp	6
○ Koncepcja	6
○ Architektura	9
▪ Architektura Systemu	9
▪ Architektura Serwerowa	11
▪ Architektura Aplikacji Klientckiej	13
▪ Architektura Bazy Danych	14
• Implementacja	18
○ Technologie	18
○ Część serwerowa	20
▪ Wstęp	20
▪ Autentykacja aplikacji	20
▪ Autentykacja użytkownika	20
▪ API	21
○ Część Klientcka	22
▪ Wstęp	22
▪ Moduły	23
• Prezentacja aplikacji	28
• Podsumowanie	43
○ Rezultat aplikacji	43
○ Plany rozwoju	44
○ Wniosek	44
• Bibliografia	45
• Wykaz rysunków	46
• Wykaz załączników	47

1 Wstęp

1.1 Cel pracy

Celem tej pracy jest zgłębienie tematyki problemu przechowywania danych, określenie specyfiki oraz określenie założeń potrzebnych do zaprojektowania systemu starającego się rozwiązać ten problem.

Poruszone zostaje także tematyka istniejących rozwiązań oraz powody, dla których rozwiązania te nie są wystarczające.

1.2 Charakterystyka problemu

W dzisiejszym świecie jesteśmy otoczeni przez coraz większą ilością danych.

Niemożliwym jest zapamiętanie ich wszystkich, tym bardziej że dane te często ulegają przedawnieniu lub modyfikacji.

Staramy się radzić sobie z tym problemem zapisując dane w różnorodnych miejscach i na różnorakie sposoby, takie jak:

- zapisywanie stron w zakładkach przeglądarki
- trzymanie kodu w repozytorium
- zapisywanie plików w odpowiedniej strukturze folderów

Rozwiązania te mają jednak następujące wady:

- trudna nawigacja (płaska struktura hierarchiczna nie odwzorowuje dobrze charakteru danych, które często są ze sobą powiązane)
- duplikacja danych
- brak możliwości łatwego przeszukiwania

Niejednokrotnie zdarza się też zapomnieć o stworzonej w przeszłości strukturze tylko po to, żeby utworzyć nową lub wyczyścić dane formatując dysk (w przypadku danych zapisanych lokalnie), co skutkuje ich utratą.

Podsumowując, istotą problemu jest brak sposobu zapisu danych który spełnia poniższe założenia

- prosta nawigacja
- wizualizacja struktury danych
- łatwe przeszukiwanie
- dowolność w łączeniu danych

- zabezpieczenie danych
- centralne przechowywanie danych

1.3 Istniejące rozwiązania

Ponieważ problem istnieje niemal tak długo jak sama informatyka, istnieje wiele rozwiązań w mniejszym lub większym stopniu adresujących ten problem.

Spśród sposobów radzenia sobie z tym problemem warto wyróżnić:

- przechowywanie danych w specyficznej strukturze przy użyciu systemu plików
- przechowywanie danych na dysku sieciowym
- przechowywanie danych w dedykowanej aplikacji

Przechowywanie danych w systemie plików

Podstawowym sposobem przechowywania danych jest przechowywanie ich w odpowiednio ustrukturyzowanej strukturze plików i folderów, przy pomocy systemu plików

Rozwiązanie to pomimo swojej prostoty ma jednak swoje wady.

Po pierwsze struktura systemu plików jest strukturą drzewa, oznacza to że nie oddaje w pełni charakteru danych, które są między sobą powiązane. To z kolei wymusza na użytkownika niejednokrotnie niełatwą decyzję o tym, gdzie konkretna informacja powinna się znaleźć. Powoduje to też problemy z późniejszym wyszukiwaniem informacji. Wyszukując konkretnej informacji po dłuższym czasie, użytkownik może dojść do innego wniosku niż pierwotnie, co przy złożonych strukturach danych może okazać się problematyczne.

Należy tutaj zaznaczyć, iż problem przynależności danych może być rozwiązany także poprzez ich duplikację lub skróty. Duplikacja nie jest jednak nigdy dobrym pomysłem, jako że prowadzi do desynchronizacji danych. Skróty teoretycznie są w stanie rozwiązać ten problem, jednak nie są intuicyjne w użyciu dla przeciętnego użytkownika.

Kolejną wadą tego rozwiązania jest problem wyszukiwania danych. W przypadku systemu plików nie jesteśmy w stanie przeszukiwać plików według zawartości, a nazwa pliku nie zawsze jest w stanie oddać w pełni zakres informacji w nim zawarty.

Przy przechowywaniu danych na dysku istnieje także ryzyko utraty danych spowodowane fizycznym uszkodzeniem urządzenia.

Przechowywanie danych na dysku sieciowym

Kategoria ta obejmuje zarówno tradycyjne dyski sieciowe jak i tzw. *cloud storage* [1].

Sposób ten dzieli część problemów z poprzednim rozwiązaniem, rozwiązuje jednak problem potencjalnej utraty danych w przypadku fizycznego uszkodzenia plików

Przykładami dysków sieciowych mogą być:

- Google Drive [2]
- Dropbox [3]
- One Drive [4]
- Zwyczajny dysk sieciowy (w przypadku zapewnienia mechanizmów kopii zapasowych) [5]

Przechowywanie danych w aplikacji

Istnieje bardzo dużo aplikacji starających się rozwiązać problematykę przechowywania i kategoryzacji danych. Jedną rzeczą, którą można bardzo szybko zaobserwować poszukując konkretnego rozwiązania jest fakt, że większość aplikacji jest wyspecjalizowana tzn. obejmuje pewien wąski zakres funkcjonalności pod pewien konkretny przykład użycia.

Większość rozwiązań najbardziej zbliżonych koncepcyjnie do poszukiwanego nazywana jest terminem *knowledge base* (ang. baza wiedzy). Rozwiązania te są zazwyczaj rozwiązaniami komercyjnymi przeznaczonymi dla dużych firm oraz zespołów. Potrafią też być dosyć kosztowne.

Jako przykłady można podać:

- Confluence [6]
- Azure DevOps [7]

Rozwiązania te są jednak rozwiązaniami bardziej skupionymi na koordynacji i zarządzaniu projektami. Pomimo że mają komponenty przeznaczone do przechowywania i kategoryzowania danych (mechanizmy boardów, wiki), nie są one priorytetem.

Warto także napomnieć o aplikacjach do robienia notatek, które pomimo rozbudowanej możliwości tworzenia zaawansowanych widoków w prosty i przyjemny sposób, nie udostępnia możliwości tworzenia struktur.

Programem najbardziej zbliżonym jest program Obsidian [8].

Jest to program opierający się o podobne założenia, operuje jednak w głównej mierze na plikach *markdown*. Nie jest także szczególnie intuicyjny, co może przełożyć się na początkowe odrzucenie ze strony niezaawansowanego użytkownika.

Poszukiwane rozwiązanie łączy więc cechy poszczególnych istniejących rozwiązań i upraszcza sposób użycia.

2 Projektowanie

2.1 Wstęp

Sekcja ta porusza poszczególne kwestie związane z procesem projektowania systemu spełniającego poszczególne założenia wymagane do rozwiązania problemu.

2.2 Koncepcje

Sekcja ta ma na celu przybliżenie obiektów oraz koncepcji występujących w systemie.

Widok grafowy struktury danych

Widok grafowy ma na celu ułatwienie identyfikacji powiązań oraz struktury danych.

Pomimo, że funkcjonalnie nie różni się to od klasycznego widoku, interesujące kształty, animacje oraz ciekawy interfejs użytkownika zachęca i poprawia komfort korzystania z aplikacji.

Generyczność operacji

Poszczególne akcje powinny być możliwe do wykonania w wielu miejscach aplikacji i na wiele sposobów. Ma to na celu pozwolenie użytkownikowi na korzystanie z aplikacji w taki sposób, w jaki chce i w kontekstach w których chce.

Przykładem może być tworzenie obiektów, użytkownik powinien móc:

- Utworzyć obiekt, następnie podpiąć go jako dziecko
- Utworzyć obiekt z ustawionym konkretnym rodzicem
- Bezpośrednio utworzyć obiekt jako dziecko

Pozwala to na dostosowanie odpowiedniej akcji w zależności od kontekstu, poziomu znajomości struktury danych oraz stopnia zaawansowania użytkownika.

Obiekty

Główna jednostka kategoryzacji danych. Ze względu na grafowy charakter prezentacji oraz przechowywania nazywana jest jako węzeł (ang. node).

System przewiduje parę typów obiektów:

- Root (ang. korzeń) – specjalny typ obiektu, pierwotny obiekt tworzony dla każdego użytkownika

- Kontener (ang. container) – obiekt służący jako kontener, używany głównie do organizacji struktury
- Plik (ang. file) – obiekt służący jako sposób przechowywania dodatkowej zawartości

Każdy z obiektów może zawierać także widok opisujący jego przeznaczenie, zawartość

W przypadku gdy obiekt zawiera plik, zawartość pliku może być wyświetlona (w miarę możliwości)

Obiekty mogą być połączone jedną z trzech relacji

- Rodzic
- Dziecko
- Link

Obiekt może także zawierać listę uprawnień użytkowników, pozwalającą na kontrolę nad dostępem do obiektu przez innych użytkowników.

Obiekty mogą być tagowane (zawierać listę tagów).

Użytkownicy

Użytkownicy występują w systemie w dwóch kontekstach.

Pierwszym rodzajem użytkownika jest użytkownik autentykacyjny, używany do autentykacji i zabezpieczenia danych przed nieautoryzowanym dostępem.

Drugim kontekstem jest kontekst powiązywania użytkowników. W celu udostępnienia danych innemu użytkownikowi (nadania uprawnień do obiektu) wymagana jest forma powiązywania użytkowników.

Aby użytkownik A mógł wyświetlić obiekt użytkownika B, musi utworzyć powiązanie z użytkownikiem B. Następnie użytkownik B musi nadać uprawnienia do obiektu użytkownikowi A.

Rozgraniczenie wynika to z potrzeby separacji danych wymaganych do działania aplikacji od danych autentykacyjnych.

Tagi

Tagi służą jako forma oznaczenia i wyróżnienia obiektów na potrzeby kategoryzacji i wyszukiwania.

Pozwalają na rozróżnienie obiektów o tych samych nazwach i podobnych opisach.

Widoki

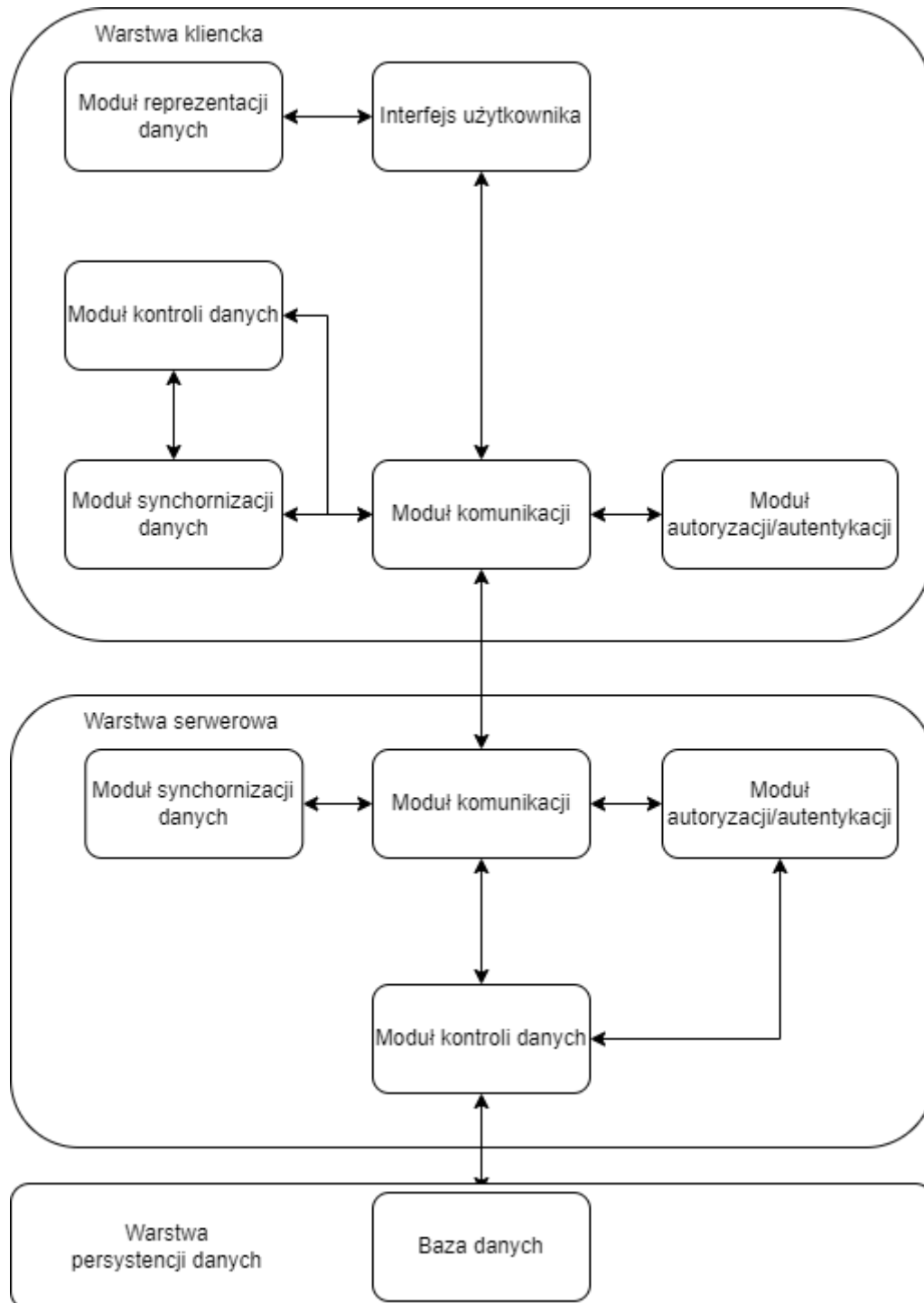
Widoki są formą wizualnego opisu zawartości obiektu. Pozwalają na dodanie dodatkowego kontekstu do niejednokrotnie niewystarczającego opisu tekstowego. Ponieważ tworzenie widoków wymaga od użytkownika znajomości języków *markup* (HTML, MD), system upraszcza i przyspiesza tworzenie widoków poprzez edytor widoków.

Edytor ten powinien umożliwiać tworzenie widoków poprzez zestaw predefiniowanych elementów.

Dodatkową zaletą takiego rozwiązania jest ujednolicenie stylów aplikacji i widoków.

2.3 Architektura

2.3.1 Architektura Systemu



[Rys. 1] Diagram przedstawiający podział systemu na moduły

Powyższy diagram architektury przedstawia logiczny podział systemu na moduły.

Bazuje on na klasycznym trójwarstwowym podziale architektury aplikacji [9] (podział na aplikację kliencką, serwer oraz bazę danych).

Taka separacja cechuje się następującymi zaletami:

- zwiększa niezawodność systemu
- ułatwia lokalizację problemów
- pozwala na łatwiejszą zamianę poszczególnych komponentów (warstwy są od siebie niezależne)

Warstwa kliencka

Moduł reprezentacji danych – odpowiedzialny za obliczenia graficzne w aplikacji np. pozycje obiektów, stan wyświetlania (wszelkie kwestie związane z graficznym podglądem struktury obiektów).

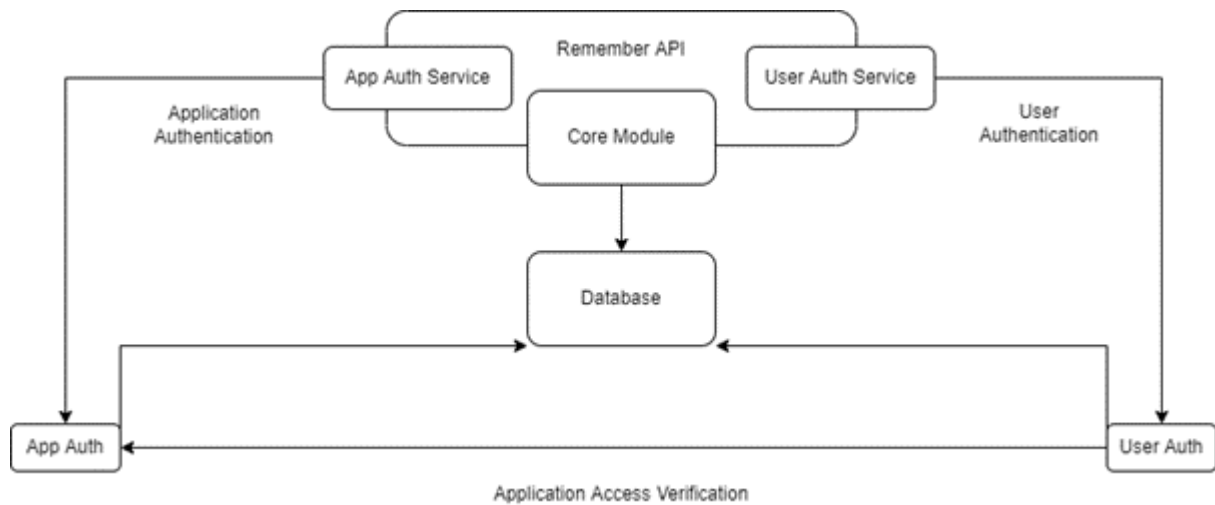
Interfejs użytkownika – pośredniczy w komunikacji użytkownika i aplikacji, propaguje akcje w dół systemu

Moduł komunikacji – odpowiedzialny za komunikację z częścią serwerową systemu, propaguje elementy potrzebne do udanej komunikacji.

Moduł autoryzacji/autentykacji – odpowiedzialny za przechowywanie oraz zarządzanie danymi sesji użytkownika

Moduł kontroli danych – odpowiedzialny za weryfikację poprawności danych, zarządzanie stanem danych i propagację akcji

2.3.2 Architektura Serwerowa



[Rys. 2] Diagram przedstawiający schemat architektury serwerowej

Powyższy diagram architektury przedstawia podział części serwerowej systemu na moduły oraz obrazuje komunikację odbywającą się pomiędzy nimi.

Widoczne są 3 główne moduły

- AppAuth – autentykacji aplikacji
- UserAuth – autentykacji użytkownika
- Remember API – główne API aplikacji

Ideą separacji jest możliwość łatwego ponownego użycia mechanizmu autentykacji użytkownika.

Moduł autentykacji aplikacji jest odpowiedzialny za autoryzację dostępu do innych modułów.

Aplikacje wyrażające chęć komunikacji z modułami (np. AppAuth) muszą się najpierw w nim zautentykować.

Wyodrębnienie moduły autentykacji użytkownika pozwala na jedną unifikowaną bazę użytkowników pomiędzy innymi modułami, usunięcie duplikacji kodu oraz potencjalnych problemów.

Separacja ta pozwala także na logiczną i potencjalnie fizyczną separację bazy danych API i poszczególnych modułów, co jest dobrym pomysłem ze względu na wrażliwy charakter danych autentykacyjnych.

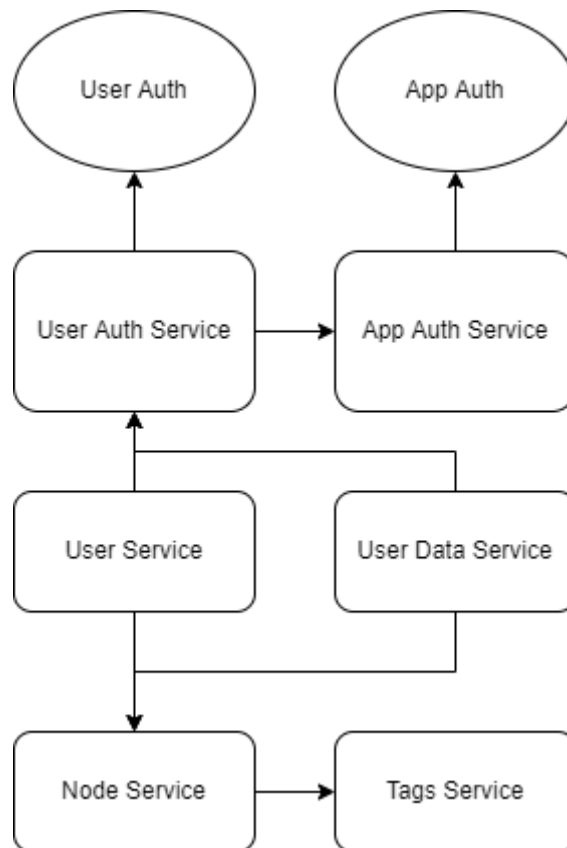
Autoryzacja aplikacji

Aplikacja/moduł komunikujący się powinien być bytem zaufanym (być zarejestrowany) w systemie. Proces ten może się odbywać poprzez zamieszczenie odpowiedniego wpisu w bazie danych, następnie weryfikację tożsamości poprzez sprawdzenie obecności wpisu oraz poprawności danych uwierzytelniających.

Autoryzacja użytkownika

Moduł autentykacji użytkownika przechowuje bazowe informacje potrzebne do identyfikacji użytkownika. Moduły potrzebujące dodatkowych informacji na temat użytkownika powinny rozszerzać wpis o użytkownika o dodatkowe, specyficzne dla tego modułu informacje.

Remember API



[Rys. 3] Widok podziału API na poszczególne moduły

User Auth i App Auth

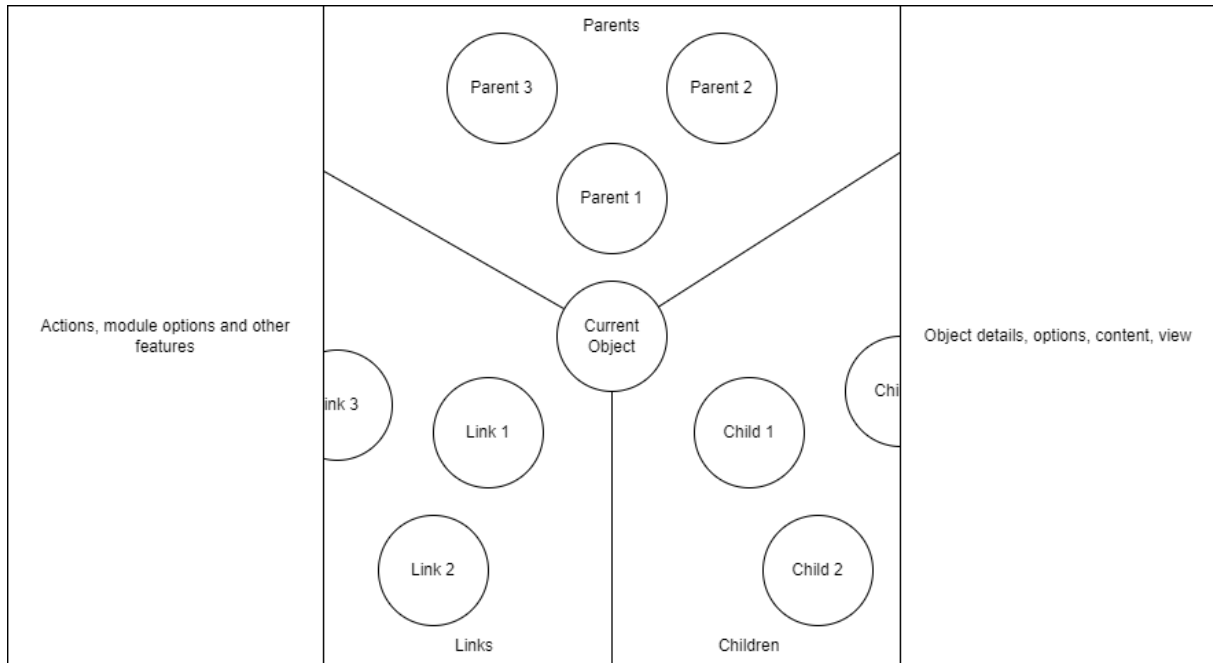
Moduły te są odpowiedzialne za komunikację z odpowiadającymi im modułami systemu.

User

Moduł odpowiedzialny za uwidocznienie interfejsu odpowiadającego za powiązania między użytkownikami. Zwraca tylko informacje o użytkownikach wymagane do działania aplikacji.

2.3.3 Architektura Aplikacji Klientkiej

Motywem przewodnim i zarazem głównym widokiem aplikacji jest poniższy schemat.



[Rys. 4] Mock głównego widoku aplikacji

Widoczne są trzy sekcje widoku:

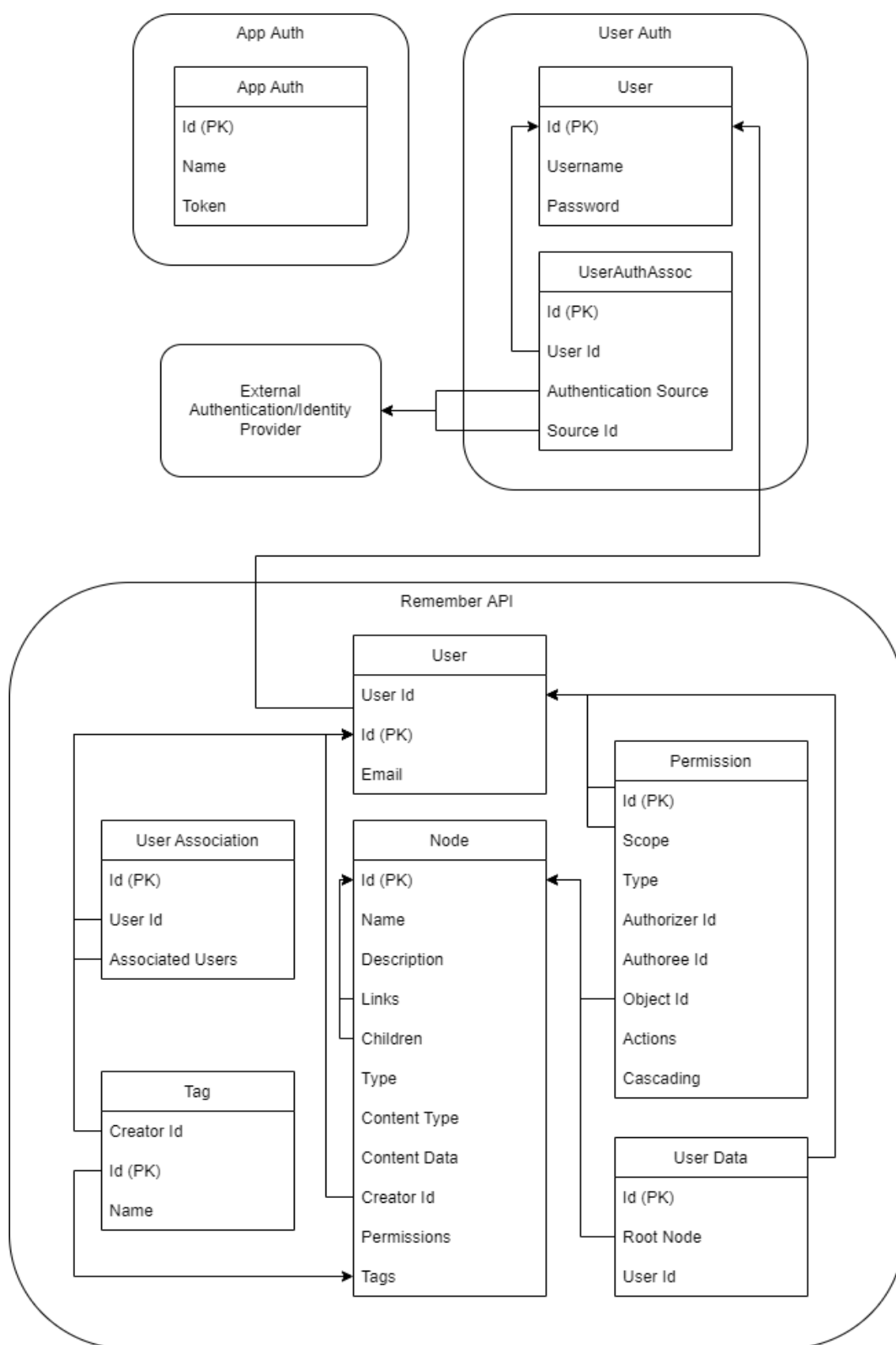
- Menu modułu (po lewej stronie)
- Widok grafowy obiektów (pośrodku)
- Widok zawartości obiektu (po prawej stronie)

Menu modułu obejmuje wszelkie operacje i informacje wykonywane w kontekście całej aplikacji.

Widok grafowy pozwala na przesuwanie, przybliżanie oraz nawigowanie obiektów.

Widok zawartości obiektu zawiera jego informacje, podgląd zawartości, widoku, oraz akcje wykonywane w jego kontekście.

2.3.4 Architektura Bazy Danych



[Rys 5] Schemat Bazy Danych

Powyższy diagram przedstawia modele danych występujące w systemie oraz powiązania między nimi.

App Auth

Id – klucz główny

Name – nazwa aplikacji

Token – generowany token służący jako poświadczenie do potwierdzenia tożsamości aplikacji

User Auth (Autoryzacja użytkownika)

Id – klucz główny użytkownika

Username – nazwa użytkownika

Password – hasło użytkownika

User Auth Assoc (Asocjacja autentykacji użytkownika)

Id – klucz główny asocjacji

User Id – klucz obcy użytkownika

Authentication Source – źródło autentykacji

Source Id – klucz obcy użytkownika w źródle autentykacji

User

Id – klucz główny użytkownika

User Id – id użytkownika autoryzacyjnego

Email – email użytkownika

User Association

Id – klucz główny asocjacji

User Id – klucz obcy, id użytkownika

Associated Users – kolekcja id użytkowników

Tag

Id – klucz główny tagu

Creator Id – klucz obcy, id użytkownika

Name – nazwa tagu

User Data

Id – klucz główny danych użytkownika

Root Node – id obiektu głównego użytkownika

User Id – klucz obcy, id użytkownika

Permission

Id – klucz główny wpisu uprawnień

Scope – zakres uprawnień

Type – typ uprawnienia (zezwól, zabroń)

Authorizer Id – klucz obcy, id użytkownika tworzącego uprawnienie

Authoree Id – klucz obcy, id użytkownika, którego dotyczy uprawnienie

Actions – lista akcji obejmująca uprawnienie

Cascading – flaga decydująca o kaskadowaniu uprawnień do obiektów dzieci

Node

Id – klucz główny obiektu

Name – nazwa obiektu

Description – opis obiektu

Links – lista obiektów linków

Children – lista obiektów dzieci

Type – typ obiektu

Content Type – rodzaj zawartości obiektu

Content Data – dane zawartości obiektu

Creator Id – klucz obcy, id użytkownika, który stworzył obiekt

Permissions – lista uprawnień podpiętych pod obiekt

Tags – lista tagów podpiętych pod obiekt

3. Implementacja

3.1 Technologie

Głównymi czynnikami decydującymi o wyborze technologii były:

- Ilość dostępnych materiałów oraz rozmiar społeczności
- Ilość i jakość dostępnych bibliotek (w szczególności bibliotek bezpłatnych)
- Elastyczność rozwiązania
- Szybkość implementacji oraz narzut potencjalnych zmian
- Wstępna znajomość technologii

Jako technologia serwerowa wybrany został NodeJS [10] wraz z frameworkiem NestJS [11].

Technologia NodeJS pozwala na szybką i łatwą implementację przy użyciu języka Typescript [16], który obudowuje język Javascript (jest nadzbiorem języka JS, w trakcie procesu zwanego transpilacją [12] tłumaczony jest na JS). Pozwala on na wiele możliwości personalizacji języka oraz rozwiązuje wiele problemów tradycyjnych silnie typowanych języków programowania. Środowisko NodeJS znane jest także z ogromnej ilości dostępnych bibliotek oraz dużego wsparcia ze strony społeczności, co pozwala na uniknięcie dużego narzutu pracy związanego z implementacją powtarzających się schematów.

#dodać źródło

Framework NestJS jest popularnym rozwiązaniem pozwalającym na szybkie i łatwe implementowanie aplikacji serwerowych. Rozwiązanie to przede wszystkim usuwa potrzebę implementacji klasycznie spotykanych mechanizmów. Zawiera także szereg bibliotek ułatwiających interoperację z innymi popularnymi rozwiązaniami oraz umożliwia użytkownikowi modyfikację jego zachowań bez potrzeby modyfikowania kodu źródłowego.

#dodać źródło

Dużym atutem jest także możliwość ujednolicenia bazy kodu dzięki wspólnemu językowi części serwerowej i klienckiej.

Jako technologia kliencka wybrany został *framework* Angular.

Angular jest technologią pozwalającą na pisanie złożonych, reaktywnych aplikacji przeglądarkowych.

Zapewnia wiele mechanizmów takich jak:

- wykrywanie zmian (ang. *change detection* [12]) modeli danych pozwalające na automatyczne odświeżanie widoków
- wstrzykiwanie zależności (ang. *dependency injection* [13])
- złożony system animacji
- system formularzy
- system tłumaczenia (internacjonalizacji [14])

Zapewnia także dużą ilość klas pomocniczych np. klienta http (HttpClient)

Rozwiązuje także wiele problemów związanych z tworzeniem aplikacji internetowych (różne standardy JS i implementacje języka, niezaimplementowane funkcjonalności przeglądarki).

Pozwala także na łatwą migrację na inne docelowe urządzenia przy zachowaniu wspólnej bazy kodu) przy użyciu narzędzi takich jak Electron [15].

Fakt ten redukuje ilość zewnętrznych zależności, ponieważ większość rzeczy potrzebnych do implementacji znajduje się już we *frameworku*, co bezpośrednio przekłada się na uproszczenie bazy kodu.

Warto także wspomnieć, iż frameworki NestJS i Angular mają wiele wspólnego, jako że NestJS został stworzony na podobieństwo Angulara (ta sama modularna konwencja i elastyczność). Oba korzystają także z języka Typescript i tego samego systemu paczek, co pozwala na wyodrębnienie wspólnych bibliotek ułatwiających integrację i utrzymanie (synchronizację kodu) części serwerowej i klienckiej (ang. *common packages*).

Oba pozwalają na ekstensywną i prostą modyfikację domyślnych zachowań oraz udostępniają zestaw klas znacznie upraszczających implementację podstawowych funkcjonalności.

Użyta została także biblioteka komponentów Angular Material [17].

Jest to biblioteka pisana przez ten sam zespół, który rozwija Angular, co gwarantuje dobrą integrację.

Pomimo że biblioteka ta nie zawiera dużo komponentów, wszystkie są bardzo dobrze przetestowane, zapewniają duże możliwości personalizacji oraz modyfikacji bez naruszenia ich podstawowego działania.

Serwerem bazodanowym została baza MongoDB [18], która jest bazą NoSQL [19]. Cechuje się przede wszystkim elastyczną strukturą dokumentów, która pozwala na szybkie prototypowanie bez konieczności poprawiania istniejących danych. Jest także bardzo popularnym rozwiązaniem używanym w tandemie z NodeJS (o czym świadczy choćby obecność modułu NestJS dedykowanego komunikacji z tą bazą danych).

Należy zaznaczyć, że w dostępnych opcjach znajdowała się baza grafowa, która idealnie nadawałaby się to tego przypadku użycia. Brak znajomości rozwiązania oraz limitacje czasowe sprawiły, że rozwiązanie to zostało przeniesione do planu rozwoju aplikacji.

W niektórych miejscach użyta została także biblioteka RxJS [20], która wywodzi się z paradygmatu programowania reaktywnego. Angular i NestJS korzystają pod spodem z tejże biblioteki. Biblioteka opiera się na wzorcu obserwatora

Autentykacja

Autentykacja w systemie opiera się na popularnym standardzie JWT (JSON Web Token) [21].

Standard ten definiuje sposób bezpiecznego przesyłania danych użytkownika oraz weryfikacji ich autentyczności wykorzystując algorytm *hash*'ujący do wyliczenia sumy kontrolnej z danych przy użyciu sekretu znanego tylko serwerowi. Poprzez przekazywanie tego tokenu serwer jest w stanie powiązać klienta z jego danymi oraz stwierdzić, czy to on wygenerował ten token. Pozwala to na bezstanowość aplikacji, co rozwiązuje wiele problemów związanych z utrzymywaniem sesji.



[Rys. 6] Zrzut ekranu przedstawiający przykładowy token JWT wraz z zawartością

3.2 Część Serwerowa

3.2.1 Wstęp

Część serwerowa została podzielona na trzy moduły:

- AppAuth – zwany także mikroserwisem autentykacji aplikacji
- UserAuth – zwany także mikroserwisem autentykacji użytkownika
- RememberAPI – zwane API aplikacji

Poniższe sekcje opisują pokrótce implementację każdego z modułów, poruszając co ciekawsze kwestie.

3.2.2 Autentykacja aplikacji

Mikroserwis autentykacji aplikacji powstał jako sposób zabezpieczenia mikroserwisów przed nieautoryzowanym dostępem.

Autoryzacja aplikacji opiera się na przekazaniu tokena JWT we wszystkich wiadomościach wysyłanych do mikroserwisów. Token ten aplikacje mogą otrzymać autentykując się w mikroserwisie aplikacji przy uruchomieniu.

3.2.3 Autentykacja użytkownika

Mikroserwis autentykacji użytkownika powstał jako sposób scentralizowania zarządzania użytkownikami na potrzeby obsługi wielu różnych modułów. Zabieg ten pozwala na dodawanie informacji użytkownika potrzebnych dla poszczególnych modułów bez utrzymywania osobnych baz użytkowników oraz zbędnej implementacji mechanizmów (duplikacji kodu).

Autentykacja użytkownika także opiera się o mechanizm JWT.

W zawartości każdej wiadomości wymagającej autentykacji przekazany musi być token.

Mikroserwis ten posiada także możliwości integracji z istniejącymi dostawcami autentykacji, zaimplementowana została integracja z serwisem Github [22], nie została ona jednak wykorzystana w aplikacji.

3.2.4 API

Zaimplementowane jako typowe REST API [23].

Oprócz standardowej implementacji wyróżnić można między innymi:

Mechanizm projekcji

Poszczególne encje (takie jak obiekty) posiadają niejednokrotnie więcej danych niż jest to potrzebne w konkretnych scenariuszach. Przykładem tego może być łączenie obiektów, które wymaga tylko id oraz nazwy do identyfikacji (wartości słownikowe). Pobieranie wszystkich danych jest w takim wypadku nadmierne. W związku z tym zastosowany został mechanizm projekcji pozwalający na przekazanie docelowej projekcji obiektu.

Pozwala to na wyeliminowanie duplikacji i unifikację kodu (nie trzeba dodawać osobnych punktów końcowych i obsługi dla różnych wariantów).

Rozwiązanie to rodzi także jednak pewne problemy związane z *cache*'owaniem [24] po stronie aplikacji klienckiej (opisane w części klienckiej).

Użytkownicy

Zgodnie z opisem w sekcji projektowania, użytkownicy API rozszerzają użytkowników mikroserwisu autentykacji użytkownika.

Zrealizowane jest to poprzez pobieranie danych adekwatnego użytkownika z mikroserwisu, następnie połączenie ich z odpowiadającymi mu danymi użytkownika API. W przypadku danych użytkownika aplikacji dane nie są łączone.

Parametryzacja

Wiele punktów końcowych w systemie przyjmuje zestaw opcjonalnych parametrów konfiguracyjnych pozwalających modyfikować rezultat akcji z poziomu zapytania. Jest to zabieg także mający na celu unifikować bazę kodu. Przykładem może być parametryzacja usuwania obiektu, która pozwala zdefiniować zachowanie w przypadku osieroconych obiektów dzieci (obiekty, które po usunięciu danego obiektu nie będą miały żadnych rodziców). Parametryzacja odnosi się także do mechanizmu dynamicznych filtrów budowanych po stronie klienta (nie tylko same wartości ale także typy porównań).

Propagacja danych w dół systemu

Umożliwienie poprawnego działania m. in. mechanizmów projekcji i parametryzacji bez niepotrzebnego obciążania bazy danych i API wymusza wykonywanie operacji projekcji i filtracji danych bezpośrednio w zapytaniu do bazy. Problem został rozwiązany poprzez rozbięcie modeli danych na dwie warstwy, warstwę modeli zapytań (ang. request) oraz warstwę modeli właściwości (ang.

props). Modele zapytań otrzymywane przez klienta zostają zmapowane na model właściwości, te z kolei bezpośrednio przekładają się na budowanie zapytań do bazy danych.

Obiekty

Ze względu na grafowy charakter danych oraz luźne połączenia, trzeba było podjąć decyzję dotyczącą implementacji tychże powiązań.

Rozpatrzono trzy warianty:

- Przechowywanie wszystkich relacji w obiekcie
- Przechowywanie relacji jednokierunkowych w obiekcie
- Przechowywanie relacji osobno

Zaimplementowany został wariant drugi, przechowywanie relacji jednokierunkowych ze względu na charakter bazy SQL oraz narzut związany z innymi implementacjami. Każdy obiekt zawiera informacje o swoich dzieciach i powiązaniach.

Przechowywanie wszystkich relacji w obiekcie wiązałoby się z aktualizowaniem obu obiektów, co zmienia charakter operacji z atomicznej [25] na złożoną (transakcyjną) i może prowadzić do desynchronizacji obiektów oraz znacznie utrudnia obsługę błędów.

Przechowywanie relacji osobno skutkowałoby ogromną kolekcją danych, której przeszukiwanie byłoby niewydajne w porównaniu do wybranej opcji, oraz wymagałoby zawsze dodatkowego zapytania w celu ich zwrócenia.

Problem ten w znacznym stopniu rozwiązuje wspomniana w rozdziale Technologie baza grafowa.

3.3 Część Klientka

3.3.1 Wstęp

Architektura aplikacji Angular opiera się w głównej mierze na mechanizmie wstrzykiwania zależności.

Aplikacja złożona jest z komponentów, które są powiązaniem HTML'owego szablonu z kodem.

Serwis jest klasą rejestrowaną w module, która następnie zostaje wstrzykiwana do konstruktora komponentu.

Kod powinien być specyficzny dla komponentu, wszelkie główne operacje realizowane powinny być w serwisach.

3.3.2 Moduły

Moduły przedstawione w tym punkcie nie odpowiadają bezpośrednio modułom przedstawionym w punkcie 4.1.1.

SharedModule

Moduł zawierający elementy współdzielone przez inne moduły.

Zawiera między innymi:

- dyrektywę pozwalającą na definiowanie zmiennych bezpośrednio w szablonie widoku
- komponent ładowania (ang. *loader*)
- serwis HTTP, służący jako rozszerzenie funkcjonalności oraz uproszczenie Angularowego klienta HTTP, definiuje domyślne opcje oraz pozwala na dodawanie domyślnych nagłówków wysyłanych z każdym zapytaniem
- serwis SessionStorage, ułatwiający używanie przeglądarkowego mechanizmu pamięci sesji
- serwis MatSnackBar, ułatwiający używanie komponentów Material Angular o tej samej nazwie
- transformator (zwany ang. *pipe*) filter, pozwalający na filtrowanie kolekcji bezpośrednio w szablonie widoku

MaterialModule

Moduł służący do uporządkowania importów komponentów biblioteki Angular Material

AuthModule

Moduł ten odpowiedzialny jest za zarządzanie sesją użytkownika.

Zawiera elementy takie jak:

- widoki logowania i rejestracji
- serwis AuthService, odpowiedzialny za wywołanie akcji autentykacyjnych, zapisuje otrzymany token JWT, reinicjalizuje dane w przypadku odświeżenia oraz uruchamia licznik czasu odpowiedzialny za automatyczne wylogowanie w przypadku przedawnienia sesji
- strażnik LoggedGuard (ang. *guard*) weryfikujący stan autentykacji i przekierowujący do logowania w przypadku próby wyświetlenia zawartości jej wymagającej

UserModule

Moduł odpowiedzialny za zarządzanie danymi użytkowników.

Zawiera serwis UserService, który przechowuje informacje o aktualnie zalogowanym użytkowniku oraz dociąga informację o użytkownikach powiązanych. Odpowiada także za wywoływanie akcji zarządzających powiązaniami użytkowników.

CoreModule

Moduł zawierający rdzeń aplikacji, odpowiada za globalne zarządzanie ścieżkami oraz komponenty nieprzynależące do żadnego z modułów. Wyodrębniony w celu umożliwienia dodawania kolejnych modułów w ramach tego projektu.

GrapherModule

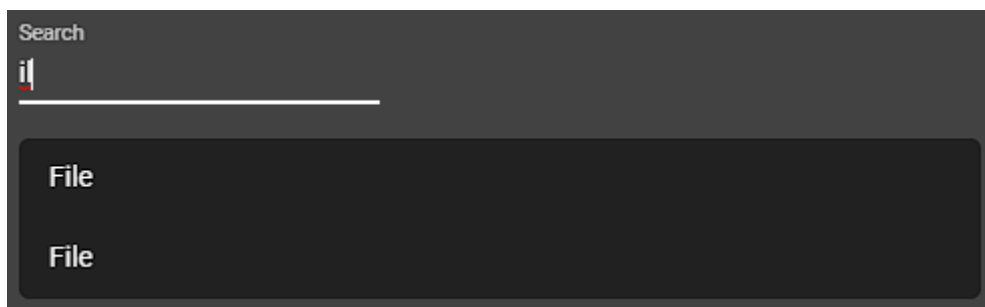
Moduł zawierający główną logikę aplikacji, nazwa wywodzi się od grafowej natury rozwiązania.

Wyodrębniony w celu późniejszego potencjalnego rozszerzenia aplikacji o inne moduły.

Komponenty

Selektor (ang. selector)

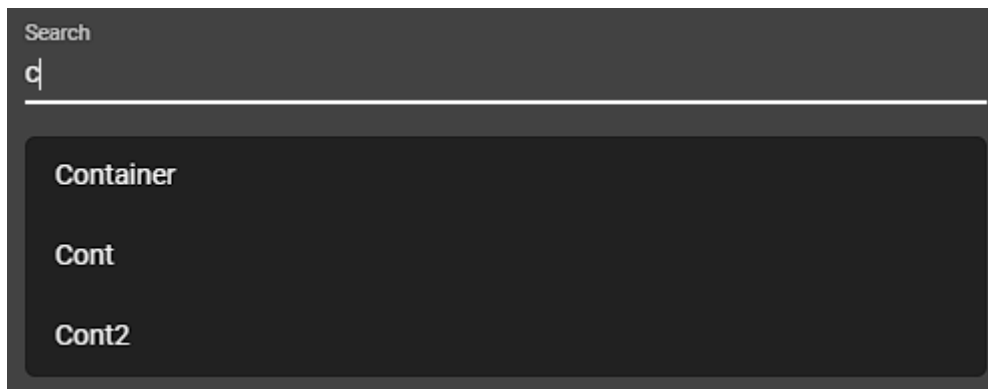
Generyczny komponent służący do wyboru pojedynczego obiektu w systemie.



[Rys. 7] Zrzut ekranu przedstawiający komponent Selektor

Linker

Generyczny komponent służący do wyboru wielu obiektów w systemie.



[Rys. 8] Zrzut ekranu przedstawiający komponent Linker

Oba obiekty wymagają dodatkowych parametryzacji związanych między innymi z wyeliminowaniem części rezultatów (np. obiekt nie powinien mieć możliwości ustawić samego siebie ani żadnego ze swoich aktualnych dzieci jako dziecka).

Okna dialogowe

Większość akcji systemu wywoływanych jest za pomocą okien dialogowych. Pozwala to na unifikację sposobu wywołania akcji z różnych miejsc w aplikacji, co przekłada się na intuicyjność i prostotę użycia.

Resizer

Generyczny komponent służący do przeskalowywania poszczególnych sekcji.



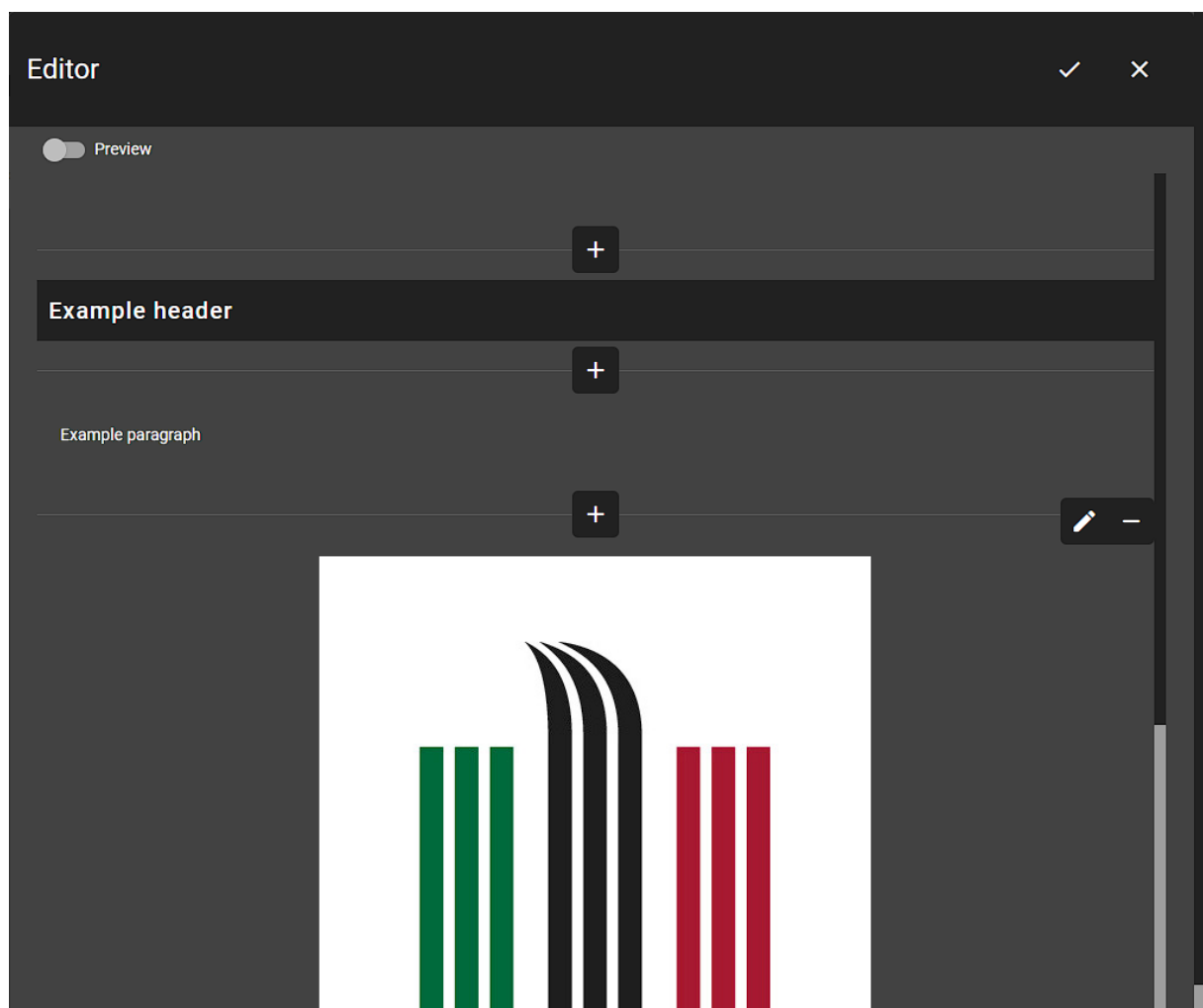
[Rys. 9] Zrzut ekranu przedstawiający komponent Resizer

Edytor (ang. editor)

Edytor widoków HTML.

Bazuje na zestawie elementów które można dowolnie dodawać do widoku, następnie edytować zawartość korzystając z właściwości contenteditable.

Potwierdzenie tworzenia widoku ekstrahuje pożądany HTML.



[Rys. 10] Zrzut ekranu przedstawiający widok edytora

Podgląd zawartości (ang. content viewer)

Komponent ten wykorzystuje natywne możliwości przeglądarki do wyświetlenia zawartości.

Klasy

Nawigator (ang. navigator)

Klasa służy do obsługi historii nawigacji po obiektach.

Cacher

Klasa pozwalająca na przechowywanie identyfikowalnych obiektów w pamięci, używana w serwisach do *cache'owania* danych.

Należy powrócić tutaj do kwestii wad projekcji poruszonych w rozdziale poświęconemu implementacji API.

Wiele projekcji tych samych obiektów powoduje problemy związane z *cache'owaniem* różnych wariantów obiektu. Rozwiązaniami tego problemu może być:

- osobne *cache'owanie* różnych wariantów
- *cache'owanie* wraz z informacją o wariancie

Docelowym rozwiązaniem jest wariant drugi, który opiera się na założeniu, że jeżeli *scache'owany* wariant obiektu jest równy lub mniejszy niż wariant pożądaný, możemy go wyliczyć, w innym wypadku musimy go pobrać. Wymaga to jednak aby poszczególne warianty były inkrementalne. Wariant pierwszy prowadzi do pobierania niepotrzebnych danych, jednak ułatwia obsługę z poziomu kodu.

Stan widoku (ang. viewer state)

Klasa służąca do przechowywania i zarządzania stanem widoku. Synchronizuje informacje w całej aplikacji.

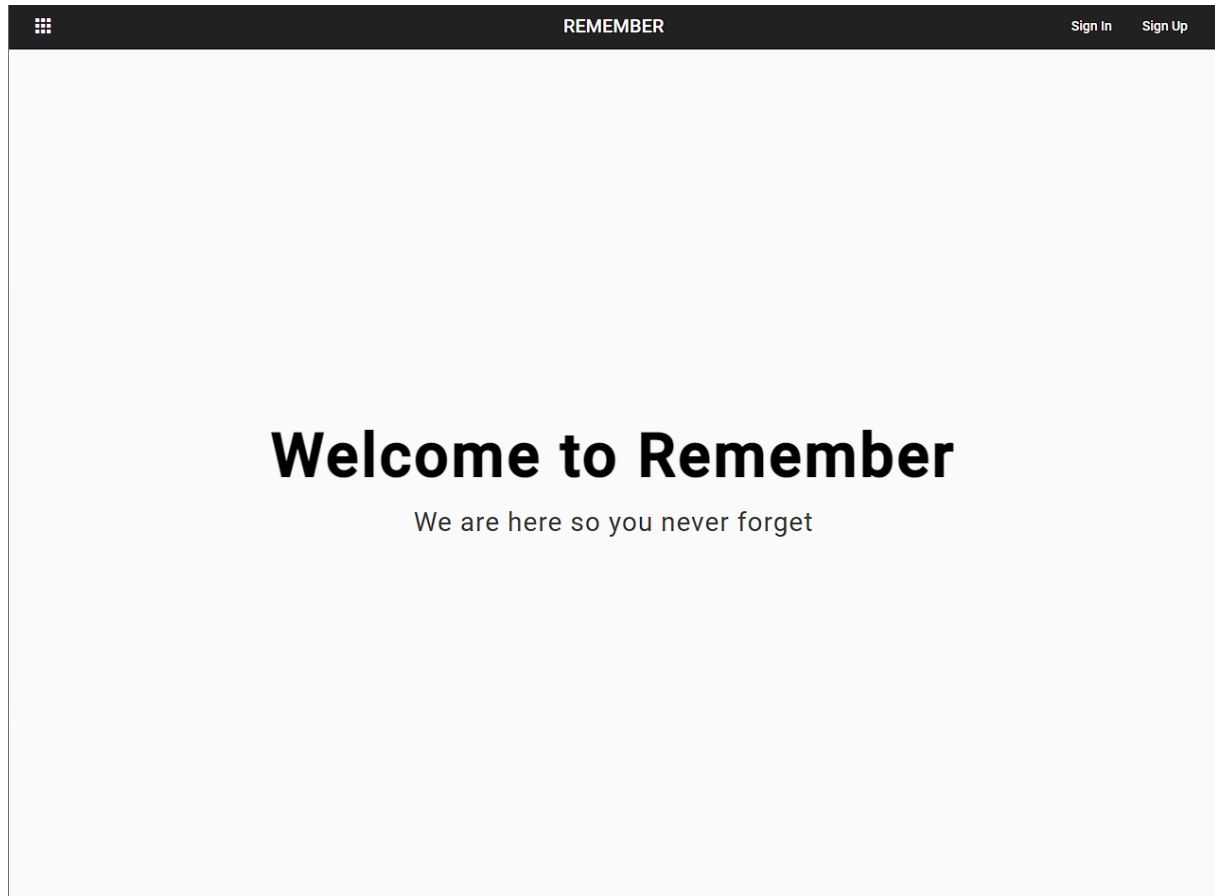
Wykorzystuje architekturę eventów do obsługi akcji z poziomu całej aplikacji (Angular EventEmitter), który bazuje na wzorcu obserwatora.

Pozwala to na wykonywanie tej samej akcji z różnymi kontekstami z dowolnego miejsca w aplikacji, a następnie spropagowanie konsekwencji tych akcji.

Rozwiązuje to problem, w którym trzy niezależne sekcje aplikacji (i ich podkomponenty) musiałyby się wzajemnie informować o zmianach.

4 Prezentacja aplikacji

Pierwszym widokiem który ukazuje się po otwarciu aplikacji jest widok powitalny



[Rys. 11] Zrzut ekranu przedstawiający widok powitalny

U góry widoczne są przyciski przekierowujące do widoków logowania i rejestracji

Sign up

Username *

Display Name *

Email *

Password *

Confirm Password *

Sign up

Already have an account? Sign in

Sign in

Login *

Password *

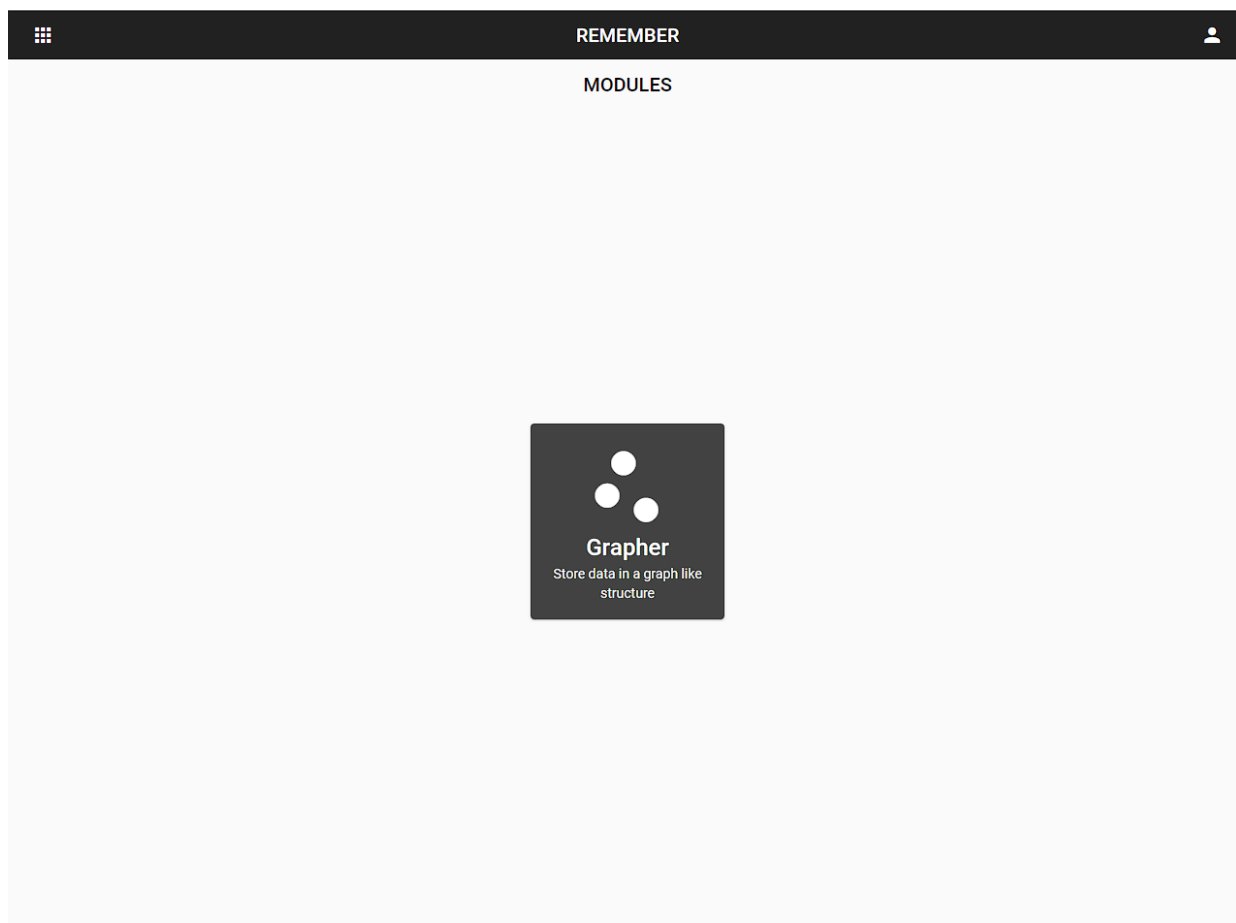
Sign in

Don't have an account? Sign up

[Rys. 12/13] Zrzuty ekranu przedstawiające widoki rejestracji i logowania (kolejno)

W przypadku podania nieprawidłowych lub niepełnych danych zostajemy o tym poinformowani.

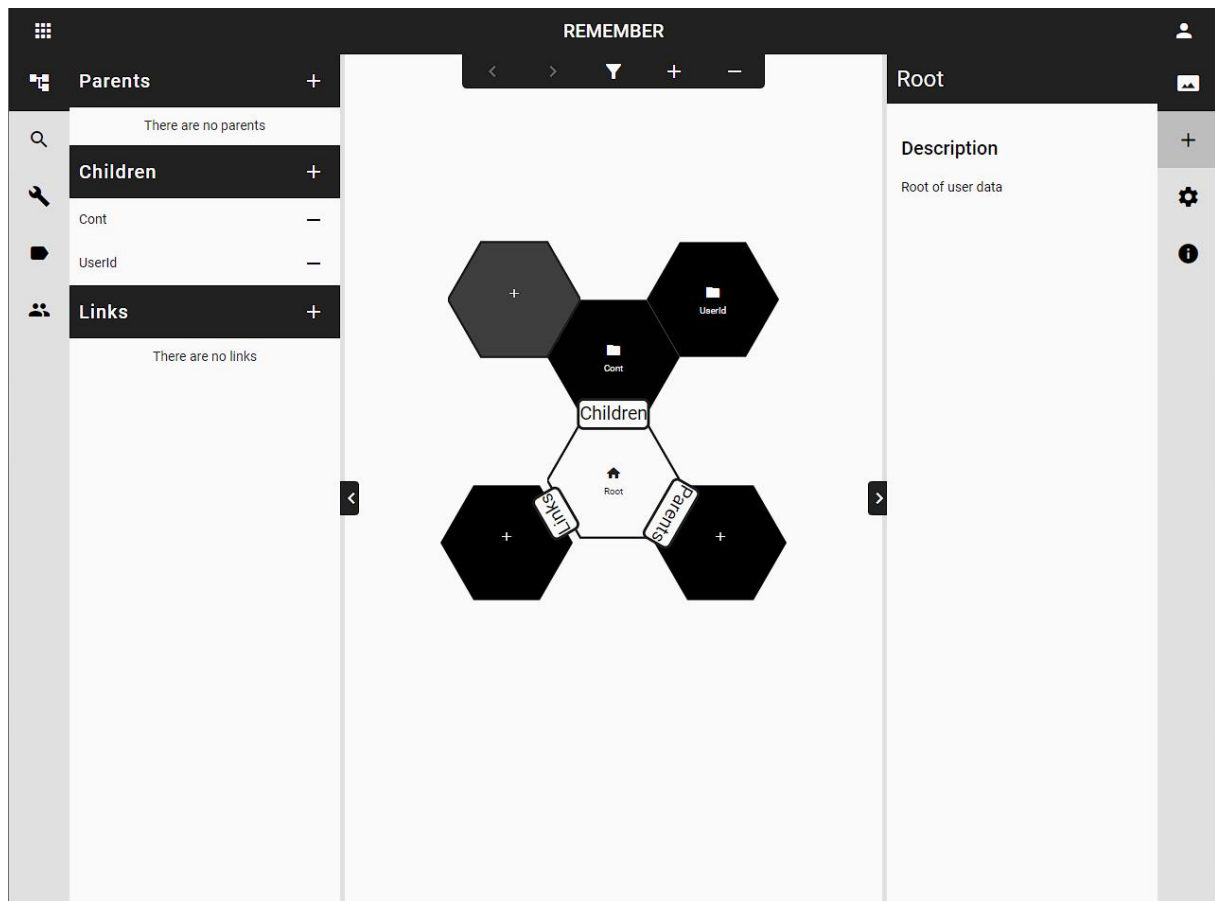
Po zalogowaniu zostajemy przekierowani do widoku modułów. To tutaj docelowo powinny znaleźć się kolejne moduły aplikacji.



[Rys. 14] Zrzut ekranu przedstawiający widok wyboru modułów

Po wybraniu modułu Grapher dochodzi do przekierowania do głównego widoku aplikacji.

Widok ten uwzględnia możliwość rozszerzenia systemu o dalsze moduły.



[Rys. 15] Zrzut ekranu przedstawiający główny widok aplikacji

Na wstępie należy wyróżnić dwa pojęcia.

Aktualny obiekt (ang. *current node*) jest to obiekt, który jest wyświetlany w środkowym polu.

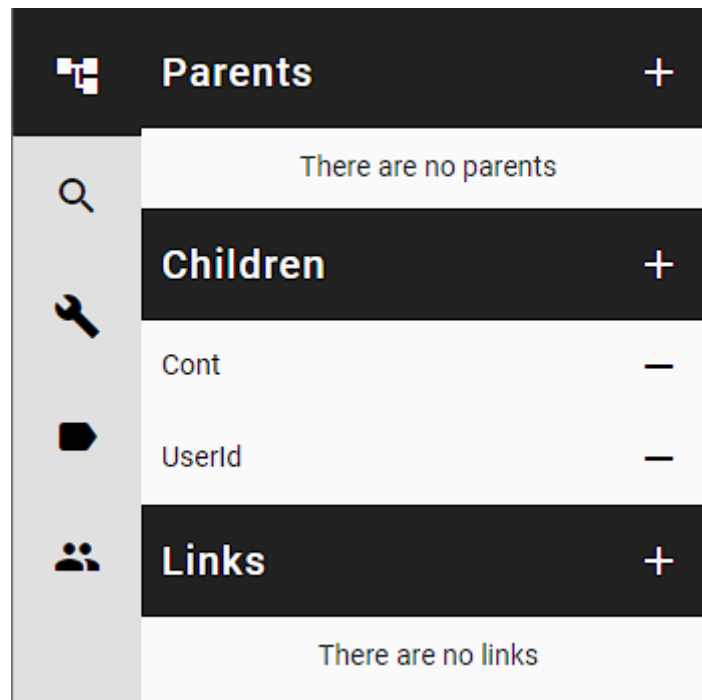
Aktywny obiekt (ang. *active node*) jest to obiekt podświetlony na biało, którego dane są aktualnie wyświetlane.

Po lewej stronie znajduje się menu modułu.

Zakładki w nim znajdujące się to:

Zakładka widoku drzewa

Zakładka ta zawiera widok drzewa dla aktualnego obiektu. Służy jako sposób klasycznego wyświetlania struktury dla użytkowników, którzy preferują klasyczny widok w stosunku do grafowego.



[Rys. 16] Zrzut ekranu przedstawiający zakładkę drzewa

Zakładka wyszukiwania

Search

Search filters:

- Name
- Description
- Type: Container
- Owner: Me
- Tag

Buttons: ✓ —

Search filters list:

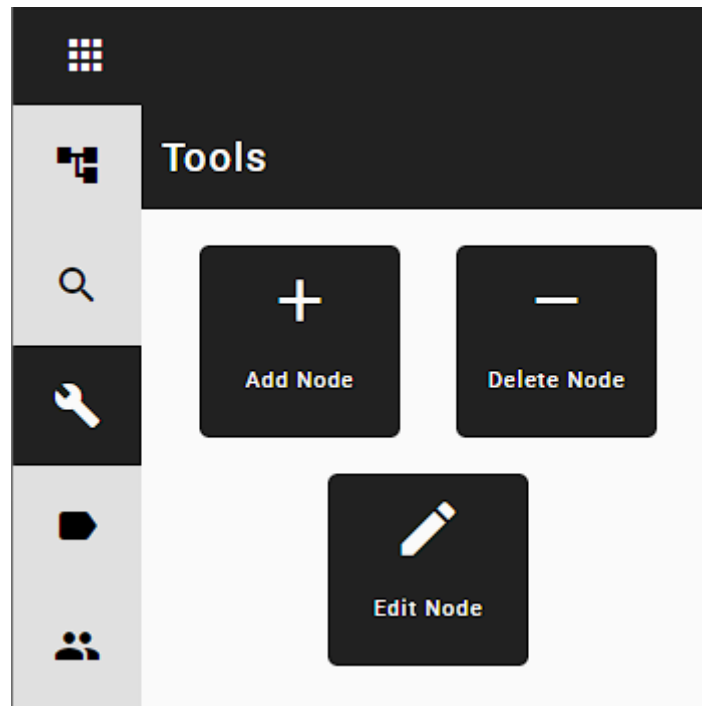
- Container
- Cont
- Cont2
- UserId

[Rys. 17] Zrzut ekranu przedstawiający zakładkę wyszukiwania

Zakładka pozwalająca na wyszukiwanie obiektów według ich właściwości (nazwy, opisu, typu, właściciela lub tagu). Pozwala także na dostęp i przeszukiwanie obiektów innych użytkowników.

Wyświetla tylko obiekty, do których użytkownik ma uprawnienia.

Zakładka narzędzi



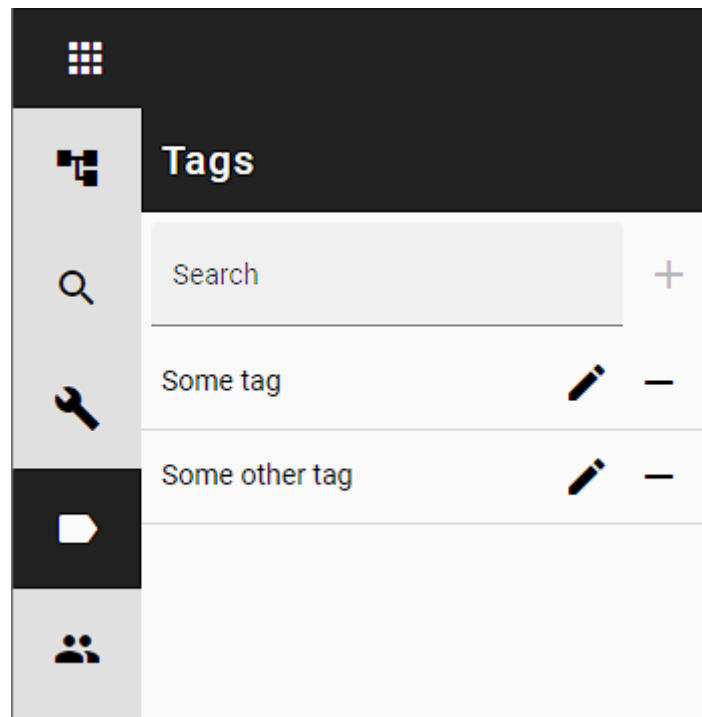
[Rys. 18] Zrzut ekranu przedstawiający zakładkę akcji modułu

Zakładka ta pozwala wywołać akcje bezkontekstowo.

Dla tworzenia oznacza to brak predefiniowanego rodzica. Oznacza to że akcja ta jest idealnym narzędziem do utworzenia obiektu w dowolnym miejscu w strukturze.

W przypadku akcji usunięcia i edycji komponenty pozwalają wybrać dowolny obiekt ze struktury.

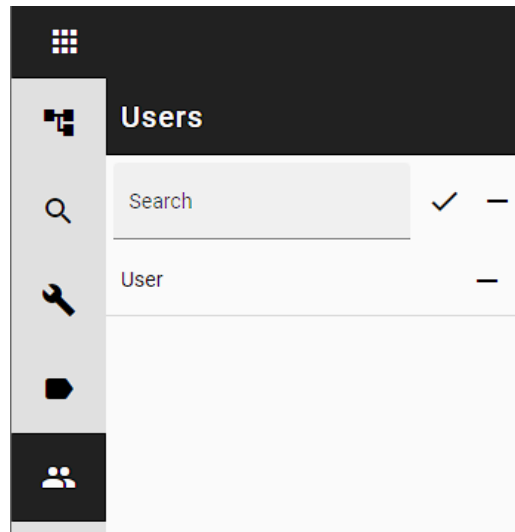
Zakładka tagów



[Rys. 19] Zrzut ekranu przedstawiający zakładkę tagów

Zakładka pozwala na zarządzanie tagami. Wylistowane są wszystkie tagi aktualnego użytkownika. Wyszukiwarka pozwala na wyfiltrowanie tagów. W przypadku, gdy tag o podanej nazwie nie istnieje, może zostać dodany.

Zakładka użytkowników



[Rys. 20] Zrzut ekranu przedstawiający zakładkę użytkowników

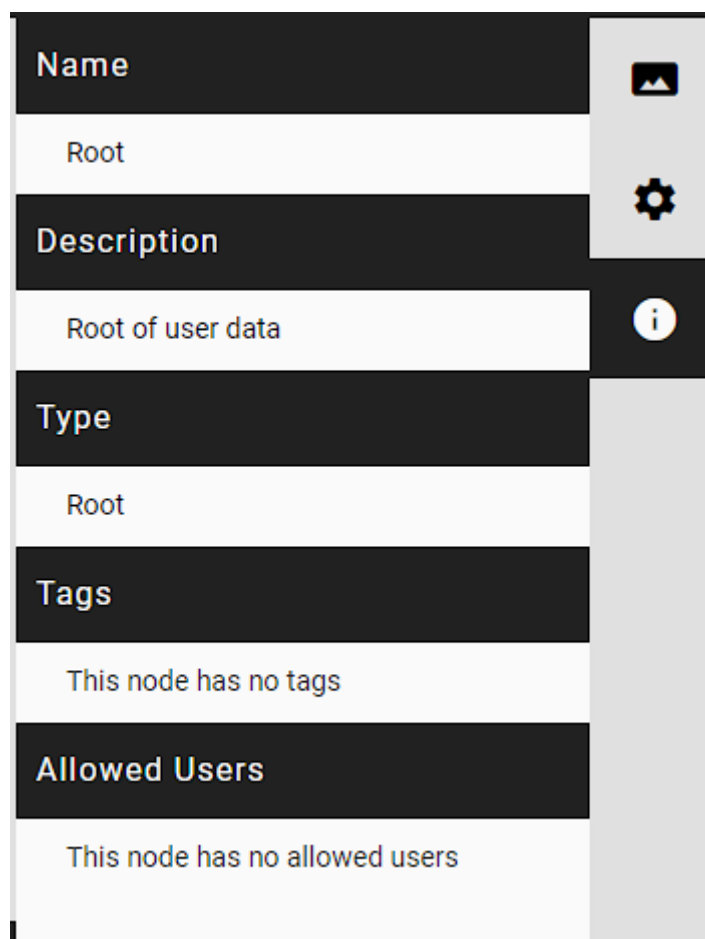
Zakładka ta pozwala na zarządzanie powiązania użytkowników.

Na liście znajdują się wszystkie aktualne powiązania. Wyszukiwarka pozwala na wyszukanie konkretnego użytkownika i dodanie asocjacji. Dodanie powiązania uwidacznia użytkownika w reszcie aplikacji.

Po prawej znajdują się menu aktywnego obiektu

Posiada takie zakładki jak:

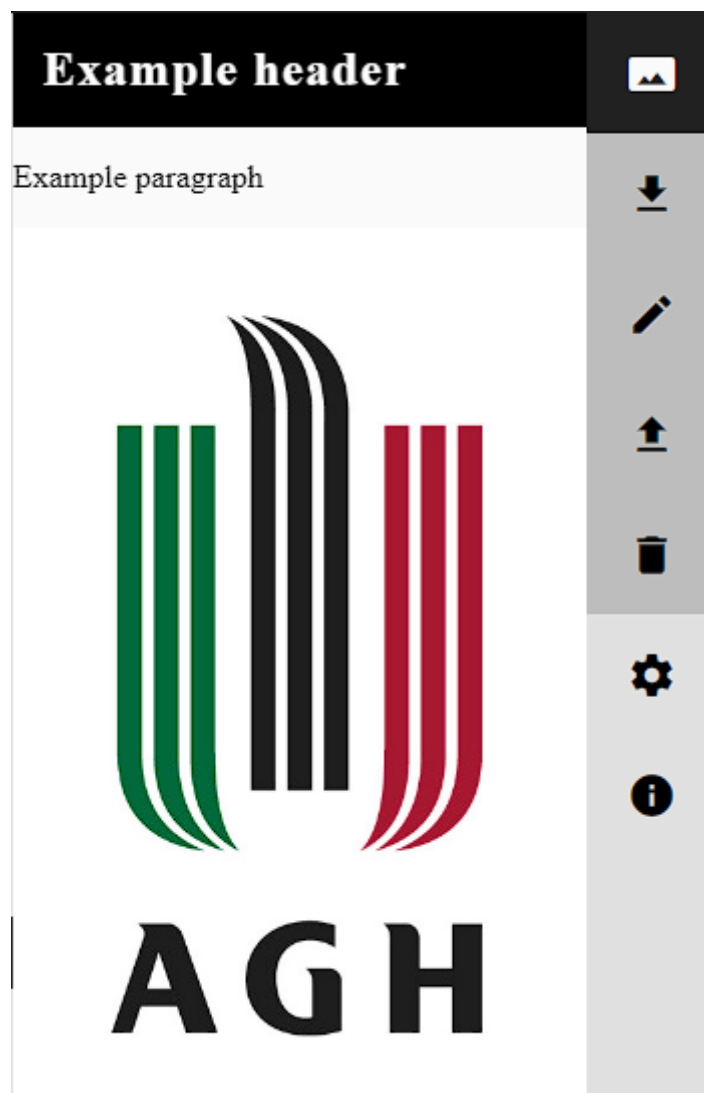
Zakładka informacji obiektu



[Rys. 21] Zrzut ekranu przedstawiający zakładkę informacji obiektu

Zakładka ta zawiera informacje na temat obiektu. W przypadku braku tagów lub uprawnień zostaje wyświetlona adekwatna sytuacja.

Zakładka widoku obiektu



[Rys. 22] Zrzut ekranu przedstawiający podgląd widoku obiektu wraz z przykładowym widokiem

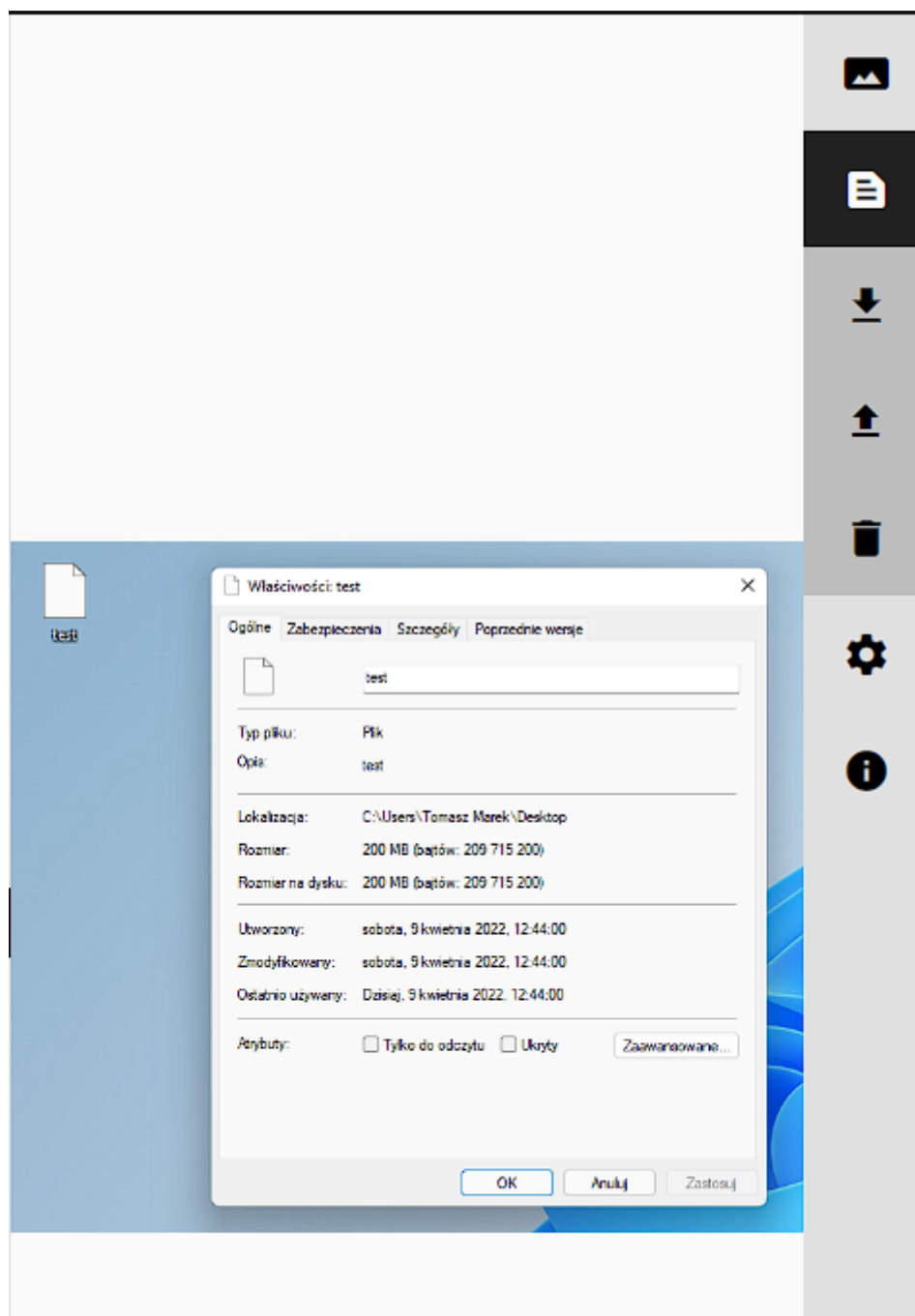
Zakładka ta pozwala na wyświetlenie widoku obiektu w przypadku jego obecności.

W sytuacji w której obiekt nie posiada widoku, wyświetlony jest domyślny widok zawierający nazwę i opis obiektu.

Po prawej stronie widoczne są akcje kontekstowe umożliwiające zarządzanie widokami:

- dodawanie widoku (tworzenie lub wgranie pliku)
- edycja widoku
- pobranie widoku
- usunięcie widoku

Zakładka zawartości obiektu



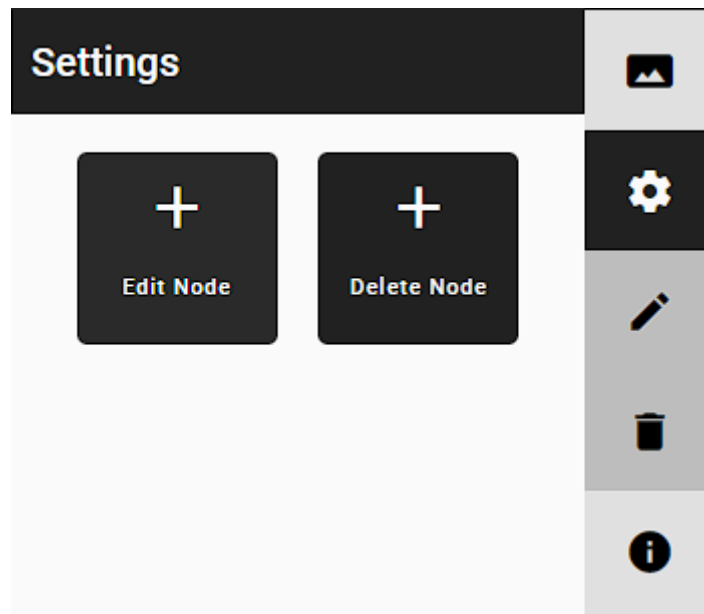
[Rys. 23] Zrzut ekranu przedstawiający zakładkę zawartości obiektu wraz z przykładową zawartością

W przypadku gdy obiekt posiada zawartość, zakładka ta pozwala na podejrzenie zawartości obiektu.

Gdy zawartość obiektu nie jest obsługiwana przez podgląd, zostaje wyświetlona adekwatna informacja.

Analogicznie, gdy obiekt nie posiada zawartości, zostaje wyświetlona informacja o jej braku.

Zakładka akcji obiektu

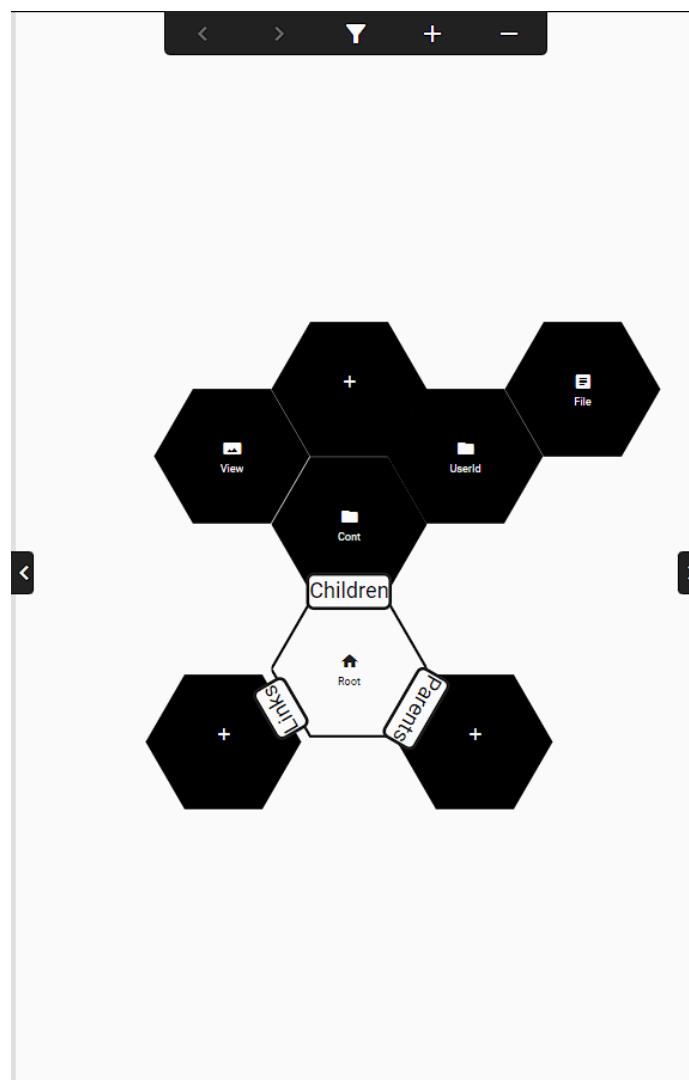


[Rys. 24] Zrzut ekranu przedstawiający widok akcji obiektu

Akcje znajdujące w tej zakładce wywoływane są w kontekście aktywnego obiektu.

Są to akcje edycji oraz usunięcia obiektu.

Widok grafowy



[Rys. 25] Zrzut ekranu przedstawiający widok grafowy obiektów

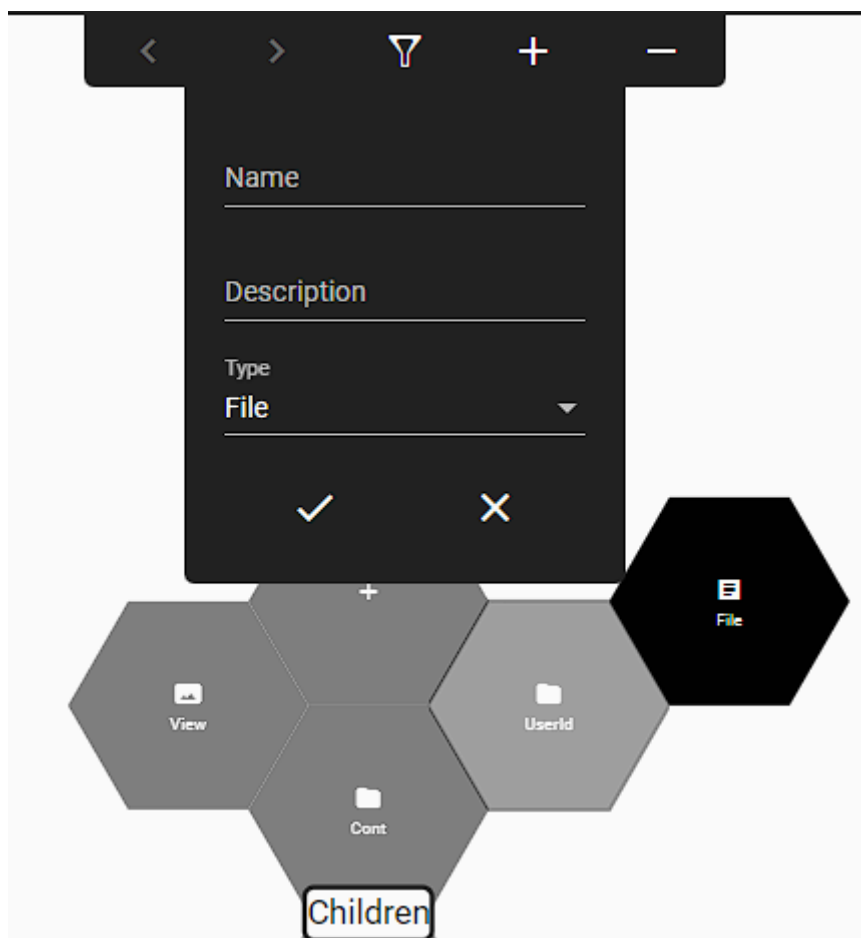
Widok ten reprezentuje aktualny obiekt wraz ze wszystkimi powiązaniem.

Aktywny obiekt charakteryzuje się białym tłem.

Obiekty rozmieszczone są na bazie koła podzielonego na trzy części.

Widok posiada możliwości zmiany przybliżenia, przesuwania oraz reskalowania.

Filtrowanie obiektów



[Rys 26] Zrzut ekranu przedstawiający filtrowanie obiektów w widoku grafowym

Jak widać na powyższym zrzucie ekranu uwidoczniony jest zestaw filtrów, umożliwiający filtrowanie obiektów na poziomie widoku.

Obiekty nie pasujące do aktywnych filtrów zostają wyszarzone. Pozwala to na szybką wizualną identyfikację pożądaných obiektów w przypadku, gdy dany obiekt zawiera dużo powiązanych obiektów (a poszukujemy konkretnego).

5 Podsumowanie

5.1 Rezultat aplikacji

Celem tej pracy było zaprojektowanie oraz implementacja systemu uniwersalnego kategoryzatora danych pozwalającego użytkownikom o różnych potrzebach i poziomach zaawansowania łatwo i wygodnie tworzyć i zarządzać strukturami danych.

Rezultatem pracy jest system spełniający wszystkie założenia wymienione w celu pracy.

System udało się zrealizować w bazowej formie, czas implementacji nie był jednak wystarczający na zaimplementowanie wszystkich docelowych funkcjonalności.

Dwoma funkcjonalnościami których nie udało się zrealizować są:

- obsługa aplikacji w trybie offline
 - synchronizacja danych pomiędzy trybem online i offline
- wersja desktopowa aplikacji

Obsługa aplikacji w trybie offline jest bardzo pracochłonna, jako że wymaga przeniesienia części logiki serwerowej do aplikacji klienckiej. Zmienia także cały proces odczytu i zapisu danych oraz wymaga analizy limitacji środowiskowych przeglądarki. Eliminuje to również możliwość mechanizmu synchronizacji danych, jako że jest on zależny od istnienia mechanizmu *offline*.

5.2 Możliwości rozwoju

Planowane usprawnienia aplikacji to:

- rozszerzenie funkcjonalności edytora
 - dodanie większej ilości dostępnych elementów
 - dopracowanie stylów widoków (w celu poprawy prezentacji widoków pobranych)
 - obsługa dodatkowych formatów (np. *Markdown*)
- dodanie obsługi wyświetlania większej ilości plików (niewspieranych przez przeglądarki)
- migracja do aplikacji desktopowej (potencjalnie mobilnej)
- migracja do bazy grafowej
- obsługa aplikacji w trybie offline
- dodanie kolejnych modułów

Użytkownicy niechętnie rejestrują się do aplikacji, jeżeli nie mają okazji jej wcześniej wypróbować, preferują też kontrolę nad tym, co powinno być zapisane w chmurze a co nie. Umożliwienie użytkownikowi z aplikacji ma na celu danie możliwości zapoznania i oswojenia się z aplikacją.

Przejsie do trybu *online* jest naturalną progresją mającą na celu zwiększenie dostępności danych użytkownika.

Migracja do aplikacji mobilnej ma na celu zwiększenie komfortu korzystania z aplikacji oraz przyspieszenie jej działania.

Kolejnymi tematycznie powiązаныmi modułami mogą być:

- Kalendarz - służący do zapisywania i przypominania o wydarzeniach i czynnościach w przyszłości, przeszłości
- Planer – służący do zapisywania czynności w formie zestawu kroków składających się na daną czynność, służy do śledzenia postępów wykonania danej czynności lub opisanie jej przebiegu (przykładami takich czynności mogą być np. plan treningowy lub przepis kuchenny)
- Notatnik – służący do spisywania szybkich pomysłów oraz notatek

Wszystkie te funkcjonalności mogą być spełnione za pomocą istniejącego rozwiązania, mogłyby jednak służyć jako interfejs upraszczający proces oraz interoperować między sobą.

Rzeczy tworzone za pomocą tych modułów mogłyby z automatu tworzyć obiekty w strukturze danych a także przeplatać się wzajemnie (współdzielić użytkowników, tagi oraz umożliwiać linkowanie obiektów).

Przykładem zastosowania mogłoby być podpięcie obiektu zawierającego harmonogram wydarzenia do tego wydarzenia w kalendarzu.

Migracja do bazy grafowej korzystnie wpłynęłaby na wydajność aplikacji.

5.3 Wniosek

Głównym wnioskiem wyniesionym z przeprowadzonych prac jest to, że nie istnieje system idealny zdolny do rozwiązania problemu kategoryzacji danych. Nie istnieje, ponieważ jest to temat zbyt zależny od dyscypliny użytkownika. System przyjazny użytkownikowi może jednak w znacznym stopniu uprościć ten proces oraz umożliwić łatwe rozwiązanie tego problemu.

Bibliografia

- [1] Artykuł opisujący magazyn w chmurze <https://aws.amazon.com/what-is-cloud-storage/> [dostęp 01.06.2022]
- [2] Oficjalna strona Google Drive, <https://www.google.pl/intl/pl/drive/> [dostęp 01.06.2022]
- [3] Oficjalna strona Dropbox, <https://www.dropbox.com/pl/> [dostęp 01.06.2022]
- [4] Oficjalna strona OneDrive, <https://www.microsoft.com/pl-pl/microsoft-365/onedrive/online-cloud-storage> [dostęp 01.06.2022]
- [5] Artykuł definiujący dysk sieciowy, <https://www.javatpoint.com/what-is-a-network-drive> [dostęp 01.06.2022]
- [6] Oficjalna strona Confluence, <https://www.atlassian.com/pl/software/confluence> [dostęp 01.06.2022]
- [7] Oficjalna strona Azure DevOps, <https://azure.microsoft.com/pl-pl/services/devops/> [dostęp 01.06.2022]
- [8] Oficjalna strona Obsidian, <https://obsidian.md/> [dostęp 01.06.2022]
- [9] Opis trójwarstwowej architektury, <https://www.ibm.com/pl-pl/cloud/learn/three-tier-architecture> [dostęp 01.06.2022]
- [10] Oficjalna strona technologii NodeJS, <https://nodejs.org/en/> [dostęp 01.06.2022]
- [11] Oficjalna strona technologii NestJS, <https://nestjs.com/> [dostęp 01.06.2022]
- [12] Opis mechanizmu wykrywania zmian Angular, <https://blog.angular-university.io/how-does-angular-2-change-detection-really-work/> [dostęp 01.06.2022]
- [13] Opis mechanizmu wstrzykiwania zależności, <https://stackify.com/dependency-injection/> [dostęp 01.06.2022]
- [14] Opis internacjonalizacji, <https://www.w3.org/International/questions/qa-i18n> [dostęp 01.06.2022]
- [15] Repozytorium projektu Electron, <https://github.com/maximegris/angular-electron> [dostęp 01.06.2022]
- [16] Oficjalna strona języka Typescript, <https://www.typescriptlang.org/> [dostęp 01.06.2022]
- [17] Oficjalna strona biblioteki komponentów Material Angular, <https://material.angular.io/> [dostęp 01.06.2022]
- [18] Oficjalna strona bazy danych MongoDB, MongoDB <https://www.mongodb.com/> [dostęp 01.06.2022]
- [19] Opis baz danych NoSQL, <https://azure.microsoft.com/pl-pl/resources/cloud-computing-dictionary/what-is-nosql-database/> [dostęp 01.06.2022]
- [20] Oficjalna strona biblioteki RxJS, <https://rxjs.dev/> [dostęp 01.06.2022]
- [21] Opis standardu JWT, <https://jwt.io/> [dostęp 01.06.2022]

[22] Oficjalna strona Github, <https://github.com/> [dostęp 01.06.2022]

[23] Opis stylu programowania REST API, <https://www.ibm.com/pl-pl/cloud/learn/rest-apis> [dostęp 01.06.2022]

[24] Opis mechanizmu pamięci podręcznej, <https://aws.amazon.com/caching/> [dostęp 01.06.2022]

[25] Definicja operacji atomowej, <https://www.techopedia.com/definition/3466/atomic-operation> [dostęp 01.06.2022]

Wykaz rysunków

Rys. 1 Diagram przedstawiający podział systemu na moduły.	9
Rys. 2 Diagram przedstawiający schemat architektury serwerowej.	11
Rys. 3 Widok podziału API na poszczególne moduły.	12
Rys. 4 Mock głównego widoku aplikacji.	13
Rys. 5 Schemat Bazy Danych.	14
Rys. 6 Zrzut ekranu przedstawiający przykładowy token JWT wraz z zawartością.	20
Rys. 7 Zrzut ekranu przedstawiający komponent Selektor.	24
Rys. 8 Zrzut ekranu przedstawiający komponent Linker.	25
Rys. 9 Zrzut ekranu przedstawiający komponent Resizer.	25
Rys. 10 Zrzut ekranu przedstawiający widok edytora.	26
Rys. 11 Zrzut ekranu przedstawiający widok powitalny.	28
Rys. 12 Zrzut ekranu przedstawiający widok rejestracji	29
Rys. 13 Zrzut ekranu przedstawiający widok logowania.	29
Rys. 14 Zrzut ekranu przedstawiający widok wyboru modułów.	30
Rys. 15 Zrzut ekranu przedstawiający główny widok aplikacji.	31
Rys. 16 Zrzut ekranu przedstawiający zakładkę drzewa.	32
Rys. 17 Zrzut ekranu przedstawiający zakładkę wyszukiwania.	33
Rys. 18 Zrzut ekranu przedstawiający zakładkę akcji modułu.	34
Rys. 19 Zrzut ekranu przedstawiający zakładkę tagów.	35
Rys. 20 Zrzut ekranu przedstawiający zakładkę użytkowników.	36
Rys. 21 Zrzut ekranu przedstawiający zakładkę informacji obiektu.	37
Rys. 22 Zrzut ekranu przedstawiający podgląd widoku obiektu wraz z przykładowym widokiem.	38
Rys. 23 Zrzut ekranu przedstawiający zakładkę zawartości obiektu wraz z przykładową zawartością.	39
Rys. 24 Zrzut ekranu przedstawiający widok akcji obiektu.	40
Rys. 25 Zrzut ekranu przedstawiający widok grafowy obiektów.	41
Rys. 26 Zrzut ekranu przedstawiający filtrowanie obiektów w widoku grafowym.	42

Wykaz załączników

1. Opis problemu, wizja i koncepcja systemu
2. Dokument specyfikacyjny
3. Dokumentacja deweloperska
4. Podręcznik użytkownika