



AGH

REMEMBER

Uniwersalny kategoryzator

Dokumentacja Deweloperska

Tomasz Marek

IEiT Informatyka niestacjonarne

2021/2022

wersja 1.6

Spis treści

• Opis projektu	2
• Część serwerowa	2
• Aplikacja webowa	19
• Baza danych	23

1. Opis projektu

1.1 Struktura

Projekt można podzielić na następujące części:

- Aplikacja webowa (kliencka)
- Część Serwerowa
- Baza danych

1.2 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 wraz z frameworkiem NestJS.

Technologia NodeJS pozwala na szybką i łatwą implementację przy użyciu języka Typescript, który obudowuje język Javascript (jest nadzbiorem języka JS, w trakcie procesu zwanego transpilacją 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*) modeli danych pozwalające na automatyczne odświeżanie widoków
- wstrzykiwanie zależności (ang. *dependency injection*)
- złożony system animacji
- system formularzy
- system tłumaczenia (internacjonalizacji)

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.

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.

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, która jest bazą NoSQL. 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 do 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, 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).

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.

2. Część serwerowa

2.1 Technologie

NodeJS

NestJs

2.2 Moduły

2.2.1 Część wspólna

Sekcja ta opisuje cechy wspólne modułów części serwerowej.

Technologie

Wybór padł tutaj na technologię NodeJS wraz z frameworkiem NestJS, który w dużym stopniu ułatwia implementację REST'owych API.

Charakterystyka języka Typescript rozwiązuje też dużo problemów spotykanych przy tradycyjnych silnie typowanych językach.

Wadą języka jest fakt, że poprawne typowanie jest w dużej mierze uzależnione od dyscypliny dewelopera piszącego kod.

Zależności

NestJS - framework API

RxJS – biblioteka

PassportJS - popularna biblioteka ułatwiająca implementację klasycznych modeli autentykacji

Swagger - popularny framework developerski uwidaczniający interfejs umożliwiający wywoływanie endpointów API oraz uwidaczniający modele przez nie używane. W dużym stopniu ułatwia szybkie testowanie zmian w trakcie prototypowania.

2.2.1 Mikroserwis Autoryzacji Aplikacji(App Auth)

Opis

Mikroserwis odpowiadający za autentykację aplikacji w innych mikroserwisach(dalej nazywany App Auth).

Autentykacja odbywa się poprzez zarejestrowanie aplikacji w mikroserwisie za pośrednictwem administracyjnego API. Dla zarejestrowanej aplikacji zostają wygenerowane ID oraz Token Aplikacji,

używane następnie do autentykacji w systemie.

Do autentykacji w innych mikroserwisach używany jest JWT Token(nazywany dalej dla rozróżnienia App Auth Token) zwracany przez App Auth po przekazaniu danych autentykacyjnych (wcześniej wspomniane ID oraz Token).

Inne Mikroserwisy znajdujące się w systemie sprawdzają otrzymywane wiadomości w poszukiwaniu Tokena Autentykacji, następnie weryfikują jego poprawność za pomocą App Auth. W przypadku braku lub niepoprawności tokena odrzucają wiadomość.

Mikroserwis ten zawiera także pomocnicze API (osobna paczka) służące jako sposób wykonywania akcji administracyjnych (takich jak rejestrowanie aplikacji) oraz paczkę common wyodrębniającą elementy wspólne obu tych paczek, nie są one jednak konieczne do działania.

Wymagania

- Baza danych MongoDB

Konfiguracja

Konfiguracja odbywa się poprzez zmienne środowiskowe (.env)

- PORT - port na którym ma nasłuchiwać aplikacja
- DB_CONN - łańcuch połączenia do bazy danych
- DB_USER - użytkownik do połączenia z bazą danych
- DB_PASS - hasło użytkownika do połączenia z bazą danych
- HOST - host, na którym ma zostać wystawiony mikroserwis (najczęściej localhost)
- SECRET - Sekret używany do enkodowania tokenów JWT

Kod

Guardy

JwtAuthGuard - Guard walidujący token JWT w wiadomości. Token powinien znajdować się w wiadomości w obiekcie auth:

```
{ "auth": { "token": "<token>" }}
```

W przypadku braku lub niepoprawności tokena wiadomość zostaje odrzucona.

Korzysta ze strategii JwtStrategy.

LocalAuthGuard - Guard autentykacyjny, przyjmuje dane autentykacyjne z wiadomości, weryfikuje je a następnie zwraca JWT token.

Korzysta ze strategii LocalStrategy.

Serwisy

AppService - główny serwis mikroserwisu

Operacje

find - zwraca informacje o aplikacji identyfikowanej podanym id

validateApp - szuka aplikacji przy pomocy id zawartego w tokenie JWT, następnie porównuje token aplikacji z tokenem zawartym w bazie.

login - generuje token JWT dla danej aplikacji

verify - weryfikuje ważność i poprawność tokena JWT

Kontrolery

AppController

Główny kontroler serwisu, uwidacznia następujące akcje:

login - zwraca token JWT dla danej aplikacji

verify - weryfikuje poprawność tokena JWT

get - zwraca informacje o zalogowanej aplikacji

2.2.2 Mikroserwis Autentykacji Użytkownika

Mikroserwis odpowiadający za autentykację użytkownika(dalej nazywany User Auth).

Centralizuje zarządzanie użytkownikami.

Nadawca wiadomości musi być zautentykowany w App Auth.

Odpowiada za tworzenie, usuwanie oraz modyfikowanie podstawowych modeli danych użytkownika.

Są to między innymi

- Hasło
- Nazwa użytkownika
- Identyfikator
- Metadane

Wymagania

- Baza danych MongoDB
- Mikroserwis Autoryzacji Aplikacji(App Auth)

Konfiguracja

- PORT - port na którym ma nasłuchiwać aplikacja
- DB_CONN - łańcuch połączenia do bazy danych
- DB_USER - użytkownik do połączenia z bazą danych
- DB_PASS - hasło użytkownika do połączenia z bazą danych
- HOST - host, na którym ma zostać wystawiony mikroserwis (najczęściej localhost)
- JWT_SECRET - sekret używany do enkodowania tokenów JWT
- JWT_EXPIRATION - czas ważności tokena JWT, zdefiniowany w sekundach (np. 6000s)

- APP_AUTH_PORT - port na którym nasłuchuje mikroserwis App Auth
- APP_AUTH_HOST - host na którym nasłuchuje mikroserwis App Auth
- GITHUB_LOGIN_URL - url logowania Github
- GITHUB_TOKEN_URL - url służący do pobrania tokena Github
- GITHUB_API_URL - url API Github
- GITHUB_CLIENT_ID - id aplikacji zarejestrowanej w Github
- GITHUB_CLIENT_SECRET - sekret aplikacji zarejestrowanej w Github

Kod

Enumeracje

AuthSourceEnum - enum określający dostawcę autentykacji/tożsamości

Guards

AppAuthGuard - sprawdza poprawność token autentykacji aplikacji w wiadomości. Jeżeli token nie został przekazany, jest niepoprawny lub aplikacja nie jest zarejestrowana, odrzuca wiadomość. Bazuje na strategii AppAuthStrategy.

JwtAuthGuard - sprawdza poprawność tokena autentykacji użytkownika w wiadomości. Jeżeli token nie został przekazany, jest niepoprawny lub użytkownik nie istnieje, odrzuca wiadomość. Bazuje na strategii JwtAuthGuard.

LocalGuard - sprawdza dane logowania zawarte w wiadomości, weryfikuje je i zwraca token JWT użytkownika. Bazuje na strategii LocalStrategy.

GithubGuard - służy do logowania użytkownika za pośrednictwem serwisu Github.

Serwisy

AuthService - serwis obsługujący komunikację z bazą danych oraz bazowe czynności autentykacyjne.

Operacje

login - generuje JWT token dla podanych danych użytkownika

getGithubToken - zwraca token bazując na danych konfiguracyjnych aplikacji Github oraz przekazanemu kodowi

getGithubUserData - zwraca dane użytkownika Github

findOrCreateByAssociation - szuka asocjacji pomiędzy użytkownikiem zewnętrznego systemu a istniejącym użytkownikiem, jeżeli znajdzie to zwraca danego użytkownika, jeżeli nie to tworzy asocjację i zwraca nowo utworzonego użytkownika bazując na danych użytkownika systemu zewnętrznego.

UserService - serwis obsługujący komunikację z bazą danych oraz bazowe czynności na użytkownikach.

Operacje

findOne - znajduje użytkownika identyfikowanego za pomocą podanego id

findByCredentials - znajduje użytkownika w oparciu o podane dane logowania

createUser - tworzy nowego użytkownika

checkNameAvailable - sprawdza dostępność danej nazwy użytkownika

AppService - główny serwis mikroserwisu

Operacje

login - zwraca token JWT dla podanego użytkownika

register - tworzy nowego użytkownika

validate user - waliduje użytkownika bazując na podanych danych logowania

find - znajduje użytkownika dla podanego id

Kontrolery

AppController - główny kontroler mikroserwisu

Operacje

2.2.3 REST API (Core API)

Zwane dalej Core API lub po prostu API, jest to serce aplikacji, odpowiada za większość logiki biznesowej.

Wymagania

- Baza danych MongoDB
- Mikroserwis Autoryzacji Aplikacji(App Auth)

Zależności

uuid - paczka do generowania unikalnych identyfikatorów

rxjs - paczka ułatwiająca programowanie reaktywne

Konfiguracja

- PORT - port na którym ma nasłuchiwać aplikacja
- DB_CONN - łańcuch połączenia do bazy danych
- APP_AUTH_SERVICE_HOST - host na którym nasłuchuje mikroserwis App Auth
- APP_AUTH_SERVICE_PORT - port na którym nasłuchuje mikroserwis App Auth
- USER_AUTH_SERVICE_HOST - host na którym nasłuchuje mikroserwis User Auth
- USER_AUTH_SERVICE_PORT - port na którym nasłuchuje mikroserwis User Auth
- APP_AUTH_ID - Id aplikacji w App Auth
- APP_AUTH_TOKEN - Token aplikacji w App Auth

Moduły

App Auth

Moduł, jak nazwa wskazuje, odpowiedzialny za komunikację z App Auth.

Zarządza zwracaniem tokenu Autoryzacji Aplikacji oraz autoryzowaniem w innych mikroserwisach.

User Auth

Moduł odpowiedzialny za komunikację z User Auth. Rozszerza bazową logikę użytkownika zawartą w User Auth o akcje kontekstowe dla aplikacji, takie jak asocjacja z adresem e-mail.

Odpowiedzialny za autentykację użytkownika aplikacji.

Core Module

Moduł zawierający logikę aplikacji, odpowiedzialny za zarządzanie obiektami takimi jak:

- Węzły (ang. node)
- Tagi
- Dane użytkownika w kontekście modułu

2.2.3.1. Kod

App Auth

Kontrolery

AuthController

Ścieżka /app/auth

Endpointy

refreshToken

Ścieżka /refresh

Wywołuje akcję odświeżenia tokena App Auth

verifyToken

Ścieżka /verify

Weryfikuje ważność tokena App Auth

Serwisy

AuthService - główny serwis modułu, przy inicjalizacji weryfikuje aplikację w App Auth, następnie udostępniając innym serwisom token autentykacyjny aplikacji

User Auth

Kontrolery

AuthController - główny kontroler modułu, odpowiada za zarządzanie autentykacją użytkownika.

Ścieżka /user

Endpointy

loginUser

Ścieżka /login

Wywołuje akcję logowania w User Auth

refresh

Ścieżka /refresh

Wywołuje akcję odświeżenia tokena w User Auth

userDetails

Ścieżka /details

Zwraca detale użytkownika z User Auth i łączy je z danymi specyficznymi dla modułu

verifyUser

Ścieżka /verify

Weryfikuje ważność tokenu autentykacji użytkownika

register

Ścieżka /register

Wywołuje akcję rejestracji użytkownika w User Auth, oraz dodaje dane użytkownika specyficzne dla modułu

Websocket Gateways

Auth Gateway - Gateway obsługujący połączenia do zarządzania sesją

Guards

JwtAuthGuard - Guard weryfikujący poprawność tokena autentykacji użytkownika w zapytaniu. Wykorzystuje strategię JwtStrategy.

Services

Auth Service - serwis odpowiedzialny za obsługę danych użytkownika specyficznych dla modułu.

Operacje

createUser - tworzy użytkownika w App Auth, a następnie dodaje dane specyficzne dla modułu powiązane z nowo utworzonym użytkownikiem.

createUserForInner - tworzy użytkownika aplikacji dla użytkownika App Auth.

findUser - zwraca użytkownika aplikacji identyfikowanego poprzez podane id.

getUsers - zwraca użytkowników aplikacji identyfikowanych poprzez id zawarte w liście podanych.

Core

Kontrolery

NodeController - kontroler obsługujący zarządzanie obiektami

Ścieżka /node

createNode

POST /

Tworzy nowy obiekt

updateNode

Patch /

Aktualizuje/edytuje obiekt

deleteNode

Delete /

Usuwa obiekt

filterNodes

Post /filter

Zwraca listę obiektów spełniających warunki filtrowania

getNodes

Post /nodes

Zwraca listę obiektów zgodnie z przekazanymi parametrami

getNodeView

Post /:id

Zwraca obiekt identyfikowany poprzez podane id

getNodeContent

Get /:id/content

Zwraca zawartość dla obiektu identyfikowanego poprzez podane id

getNodeView

Get /:id/view

Zwraca widok dla obiektu identyfikowanego poprzez podane id

deleteNodeView

Delete /:id/view

Usuwa widok dla obiektu identyfikowanego poprzez podane id

replaceNodeView

Put /:id/view

Podmienia widok dla obiektu identyfikowanego poprzez podane id

removeLinks

Delete /:id/links

Usuwa linki (powiązanie) identyfikowane poprzez podaną listę identyfikatorów

addLinks

Post /:id/links

Usuwa linki (powiązanie) identyfikowane poprzez podaną listę identyfikatorów

removeParents

Delete /:id/parents

Usuwa rodziców (powiązanie) identyfikowanych poprzez podaną listę identyfikatorów

addParents

Post /:id/parents

Usuwa rodziców (powiązanie) identyfikowanych poprzez podaną listę identyfikatorów

removeChildren

Delete /:id/children

Usuwa dzieci (powiązanie) identyfikowane poprzez podaną listę identyfikatorów

addChildren

Post /:id/children

Usuwa dzieci (powiązanie) identyfikowane poprzez podaną listę identyfikatorów

TagsController - kontroler obsługujący zarządzanie tagami

Ścieżka /tags

Endpointy

addTag

Post /

Dodaje nowy tag

deleteTag

Delete /:id

Usuwa tag identyfikowany poprzez podane id

updateTag

Patch /:id

Aktualizuje tag identyfikowany poprzez podane id

filterTags

Post /filter

Zwraca listę tagów spełniających podane warunki filtrowania

UserDataController - kontroler obsługujący zarządzanie danymi użytkownikami specyficznymi dla modułu

Ścieżka /user-data

Endpointy

getNodeContent

Ścieżka /:id

Zwraca dane użytkownika specyficzne dla modułu identyfikowane poprzez podane id użytkownika

UserController - kontroler obsługujący zarządzanie relacjami pomiędzy użytkownikami oraz zwracający informacje o użytkownikach

Ścieżka /user

Endpointy

getAssociatedUsers

Get /associated

Zwraca użytkowników powiązanych z danym użytkownikiem

addAssociation

Post /associated

Dodaje powiązanie z podanym użytkownikiem

removeAssociation

Delete /associated

Usuwa powiązanie z podanym użytkownikiem

Enumeracje

FilterOperation - enumeracja określająca rodzaj operacji filtrowania (np. Zawiera, Równe, Większe niż), używana do budowania dynamicznych zapytań.

NodeType - enumeracja określająca rodzaj obiektu (Node)

NodeContentType - enumeracja określająca rodzaj zapisu zawartości obiektu (bezpośrednio w bazie lub na dysku).

Gateways

SyncGateway - Websocketowy gateway używany do informowania klienta o wszelkich zmianach obiektów niezainicjowanych przez użytkownika lub zainicjowanych z poziomu innego klienta.

Serwisy

NodeService - serwis odpowiedzialny za zarządzanie obiektami (Node)

Akcje

createNode - akcja służąca do utworzenia nowego obiektu

createNodes - akcja służąca do utworzenia wielu nowych obiektów

createNodeTree - akcja służąca do utworzenia drzewa obiektów

createNodeTrees - akcja służąca do utworzenia wielu drzew obiektów

removeNode - akcja służąca do usunięcia obiektu

removeNodes - akcja służąca do usunięcia wielu obiektów

updateNode - akcja służąca do aktualizacji obiektu

getNodeById - akcja służąca do zwrócenia obiektu identyfikowanego poprzez podane id

getNodesById - akcja służąca do zwrócenia wielu obiektów identyfikowanych poprzez podaną listę identyfikatorów

filterNodes - akcja służąca do filtrowania obiektów w oparciu o podane warunki filtrowania

getNodeContent - akcja służąca do zwrócenia zawartości obiektu

getNodeView - akcja służąca do zwrócenia widoku obiektu

deleteNodeView - akcja służąca do usunięcia widoku obiektu

replaceNodeView - akcja służąca do podmiany widoku obiektu

addLinks - akcja służąca do dodania linków (powiązań) obiektu

removeLinks - akcja służąca do usunięcia linków (powiązań) obiektu

addParents - akcja służąca do dodania rodziców (powiązań) obiektu

removeParents - akcja służąca do usunięcia rodziców (powiązań) obiektu

addChildren - akcja służąca do dodania dzieci (powiązań) obiektu

removeChildren - akcja służąca do usunięcia dzieci (powiązań) obiektu

Widoki zostały zaimplementowane jako samodzielne obiekty (Node'y) przyłączone jako dzieci do docelowego obiektu.

Akcje działające na mnogich obiektach mają na celu optymalizację oraz ograniczenie ilości zapytań, są jednak problematyczne w kontekście tworzenia wielu obiektów zawierających zawartość (ograniczenia związane z limitem rozmiaru przesyłanego zapytania).

Większość operacji w kontekście obiektów jest subsetem bazowych operacji utwórz/edytuj/usuń obiekt (tak jak na przykład operacje widoków, powiązania). Zostały jednak wyodrębnione w celu klarowności oraz łatwości developmentu, docelowo powinny jednak zostać obsłużone jako wywołania jednej bazowej funkcji jako jedna ścieżka wykonania z odpowiednią wstępną parametryzacją).

TagsService - serwis odpowiedzialny za zarządzanie tagami

Akcje

getTags - akcja służąca do zwrócenia tagów identyfikowanych poprzez podaną listę identyfikatorów

getTag - akcja służąca do zwrócenia tagu identyfikowanego poprzez podane id

addTag - akcja służąca do stworzenia nowego tagu

addTags - akcja służąca do stworzenia wielu nowych tagów

deleteTag - akcja służąca do usunięcia tagów

updateTag - akcja służąca do aktualizacji tagu

filterTags - akcja zwracająca tagi spełniające podane warunki filtrowania

UserDataService - serwis odpowiedzialny za zarządzanie danymi użytkownika specyficznymi dla modułu

Akcje

getUserData - zwraca dane użytkownika, w przypadku ich braku tworząc je

createNewUserData - inicjuje dane użytkownika dla modułu

UserService - serwis odpowiedzialny za zarządzanie relacjami między użytkownikami oraz zwracanie danych użytkowników

Akcje

getUserAssociations - zwraca powiązania dla danego użytkownika

addUserAssociation - dodaje powiązanie dla podanego użytkownika

addUserAssociations - dodaje powiązanie dla podanych użytkowników

removeUserAssociation - usuwa powiązanie dla podanego użytkownika

removeUserAssociations - usuwa powiązanie dla podanych użytkowników

getAssociatedUsers - zwraca użytkowników powiązanych z danym użytkownikiem

3. Aplikacja webowa (kliencka)

3.1 Technologie

Angular

Angular Material

3.2 Moduły

3.2.1 Autoryzacja (Auth)

Moduł odpowiedzialny za zarządzanie sesją użytkownika w aplikacji.

Kod

Guards

LoggedGuard - blokuje dostęp niezalogowanych użytkowników do widoków wymagających autentykacji

Klasy

Timer - klasa odpowiedzialna za automatyczne wylogowanie i odświeżanie sesji po stronie klienta

Serwisy

AuthService - serwis odpowiedzialny za autentykację oraz zarządzanie stanem sesji

SocketService - serwis odpowiedzialny za zarządzanie stanem sesji oraz dynamiczne aktualizowanie jej stanu w wypadku akcji niezapoczątkowanych przez klienta lub z poziomu innego klienta

Widoki

Login - widok logowania użytkownika

Register - widok rejestracji użytkownika

3.2.2 Współdzielony (Shared)

Moduł zawierający elementy (serwisy, pipe'y i widoki) wspólne dla wielu modułów.

Dyrektywy

NgVar - dyrektywa pomocnicza pozwalająca na tworzenie zmiennych pomocniczych bezpośrednio w widokach

Pipe'y

Filter - pipe filtrujący kolekcję danych według przekazanego predykatu

SafeUrl - pipe sanityzujący url w celu możliwości zastosowania go w widoku

Serwisy

Cookies - serwis pomagający w obsłudze ciastek

Http - serwis ułatwiający komunikację za pomocą klienta http

SnackBar - serwis ułatwiający wyświetlanie notyfikacji

LocalStorage - serwis ułatwiający korzystanie z lokalnej pamięci przeglądarki

Widoki

Loader - spinner używany do sygnalizacji ładowania

3.2.3 Material

Moduł utility odpowiedzialny za importowanie wymaganych modułów Angular Material.

3.2.4 Core

Moduł odpowiedzialny za podstawową funkcjonalność aplikacji (szablon).

Elementy takie jak:

- strona główna
- nawigacja na poziomie aplikacji
- zarządzanie kontem
- widok modułów

Widoki

ModuleHub - widok zawierający listę dostępnych modułów

Home - strona główna aplikacji

MainToolbar - pasek główny aplikacji zawierający akcje kontekstowe aplikacji, panel użytkownika, ustawienia.

3.2.5 Grapher (kategoryzator grafowy)

Moduł odpowiedzialny za kategoryzator grafowy.

Zawiera logikę zarządzania węzłów.

Klasy

Cacher - klasa odpowiedzialna za cache'owanie obiektów, używana do optymalizacji pobierania poszczególnych encji w systemie

Navigator - klasa odpowiedzialna za przechowywanie historii nawigacji

ViewerState - klasa odpowiedzialna za stan wyświetlania modułu, między innymi za aktualnie wyświetlane obiekty, aktywne obiekty, propagację eventów, aktualizację stanu w oparciu o zdarzenia.

Node - klasa odpowiedzialna za obsługę i parsowanie obiektów. Zawiera operacje parsowania rezultatu zwracanego przez API do klasy używanej w logice frontu.

Enumeracje

NodeAction - enum określający rodzaj operacji/zdarzenia związanego z obiektem, pozwala na scentralizowanie obsługi tej samej akcji w wielu miejscach aplikacji i w różnych kontekstach.

NodeSection - enum określający poszczególne sekcje obiektu, tj. rodziców, dzieci oraz linki.

Serwisy

Node - serwis zarządzający wywołania akcji API związanych z obiektami

Socket - serwis zarządzający połączeniem WebSocket z Gateway'em synchronizacji, aktualizuje stan aplikacji zgodnie z otrzymywanymi wiadomości

Tag - serwis zarządzający tagami

UserData - serwis zarządzający danymi użytkownika w kontekście modułu

Widoki/komponenty

ContentViewer - widok służący do wyświetlenia detali obiektu

NodeContent - widok służący do wyświetlania zawartości (pliku) obiektu jeśli takowa istnieje, z uwzględnieniem jej typu, m.in.

HtmlViewer

PlainTextViewer

UnknownViewer

GraphMenu - widok służący do wyświetlenia menu obiektów

GraphMenuSearch - widok służący do wyszukiwania, podglądania i nawigowania do obiektów

GraphMenuTags - widok służący do wyświetlania i zarządzania tagami

GraphMenuTools - widok służący do wyświetlania narzędzi i akcji dla obiektów

GraphMenuTree - widok służący do wyświetlania drzewa aktualnie wybranego obiektu

GraphViewer - widok służący do wyświetlania wizualnej reprezentacji struktury grafowej obiektu

GraphViewerNode - komponent służący do wyświetlenia węzła dla pojedynczego obiektu

Linker - bazowy komponent służący do linkowania i wyszukiwania generycznych obiektów

NodeLinker - komponent służący do linkowania i wyszukiwania obiektów (Node)

TagLinker - komponent służący do linkowania, wyszukiwania i dodawania tagów

Editor - widok pozwalający na tworzenie widoków html

4 Baza danych

4.1 Technologie

MongoDB

4.2 Zależności

Mongoose wraz z paczką dedykowaną NestJS

4.3 Opis

Baza danych (serwer) jest współdzielona pomiędzy mikroserwisami i API.

Nazwy baz dla poszczególnych modułów:

App Auth - app-auth

Kolekcje

appauths

Modele

AppAuth

_id: ObjectId - Primary key, autogenerowany

name: string - Nazwa aplikacji

token: string - Token aplikacji

User Auth - user-auth

Kolekcje

userauthassoc

users

Modele

UserAuthAssoc

_id: ObjectId - Primary key, autogenerowany

authSource: number - Enum odpowiadający dostawcy autentykacji/tożsamości

srcId: string - Id użytkownika w systemie dostawcy

userId: string - Id użytkownika User

User

_id: ObjectId - Primary key, autogenerowany

username: string - Nazwa użytkownika

password: string - Hasło użytkownika

API - remember

Kolekcje

users

userdatas

userassocs

tags

nodes

Modele

User

_id: ObjectId - Primary key, autogenerowany

userId: string - Id użytkownika w User Auth

email: string - Email użytkownika

UserData

_id: ObjectId - Primary key, autogenerowany

rootNode: string - Id obiektu (Node)

userId: string - Id użytkownika (User)

UserAssociation

_id: ObjectId - Primary key, autogenerowany

userId: string - Id użytkownika (User)

associated: string[] - Lista Id użytkowników powiązanych (User)

Node

_id: ObjectId - Primary key, autogenerowany

name: string - Nazwa obiektu

description: string - Opis obiektu

nodeLinks: string[] - Lista Id obiektów powiązanych

nodeChildren: string[] - Lista Id obiektów dzieci

type: NodeType - Enum określający typ obiektu

contentType: NodeContentType - Enum określający typ zawartości obiektu

contentData: NodeContentData - Obiekt określający detale zawartości obiektu

creatorId: string - Id twórcy obiektu (User)

permissions: string[] - Lista Id użytkowników mających dostęp do obiektu (User)

tags: string[] - Lista Id tagów przypiętych do obiektu

Permission

_id: ObjectId - Primary key, autogenerowany

permissionScope: Scope - Enum określający zakres uprawnienia

permissionType: Type - Enum określający typ uprawnienia

authorizerId: string - Id osoby autoryzującej (tworzącej uprawnienie)

authoreId: string - Id osoby autoryzowanej

actions: Action[] - Enum określający jakie akcje obejmuje dane uprawnienie

cascade: boolean - Wartość określająca, czy dzieci dziedziczą uprawnienie po tym obiekcie