Uwagi

1: Dodałbym jeszcze jakieś przykłady poza GAFAM.
1: Celem niniejszej pracy dyplomowej jest zaprojektowanie i implementacja ...
3: Proszę sprawdzić, czy «back-end» jest zawsze zapisywany w ten sam sposób. Chodzi o to, żebyśmy mieli jeden konsekwentnie stosowany zapis.
3: Czy tylko hasło, czy coś jeszcze jest daną wrażliwą?
3: Czy istnieje jakiś synonim? «Przeładowanie» można różnie rozumieć.
3: Tak samo jak z back-endem, proszę sprawdzić, czy mamy zawsze jeden zapis.
4: Czy istnieją jakieś rozwiązania poza GAFAM?
7: (Admin)
7: rys.
7: (User)
7: rys.
7: (Anonymous)
11: jaka wersja?
12: przecinek
13: bez przecinka
15: Tutaj trzeba napisać kilka słów wprowadzenia / wstępu. Teraz zaczynamy od razu z grubej rury.
16: «Optymalny» to synonim słowa «najlepszy». Mamy zatem «... jednak są mniej najlepsze ...».
16: Jaki to diagram?
16: na
16: Rys. 5.2: Jaki to diagram?
17: bez przecinka
17: To nie jest najważniejsza rzecz na świecie. Nie będę o to kruszył kopii. Czy nie mamy tutaj pewnej niekonsekwencji ortograficznej? Mianowicie mamy «front-end» z myślnikiem, ale «endpoint» bez myślnika.
20: Chyba minted trochę się gubi w przypadku JSON-a.
27: ← w minted

Politechnika
Śląska

PROJEKT INŻYNIERSKI

Webowy organizer notatek, wydarzeń i zadań

Zofia LORENC
Nr albumu: 300306

Kierunek: Informatyka
Specjalność: Bazy Danych i Inżynieria Systemów

PROWADZĄCY PRACĘ
dr hab. inż. Krzysztof Simiński
KATEDRA ALGORYTMIKI I OPROGRAMOWANIA
Wydział Automatyki, Elektroniki i Informatyki

Gliwice 2024

Tytuł pracy

Webowy organizer notatek, wydarzeń i zadań

Streszczenie

Organizer to wykorzystująca bazę danych aplikacja webowa pozwalająca na tworzenie i udostępnianie zalogowanym użytkownikom plików: notatek, wydarzeń i zadań oraz dodawanie do nich powiadomień. Ponadto ułatwia ona umawianie terminów wydarzeń dzięki funkcjonalności głosowania.

Słowa kluczowe

Angular, MySQL, Spring, Organizer

Thesis title

Web organizer of notes, events, and tasks

Abstract

Organizer is a web application that uses database and allows logged users to create and share files: events, notes and tasks, and add reminders for them. Moreover, it helps deciding on event dates thanks to its voting functionality.

Key words

Angular, MySQL, Spring, Organizer

Spis treści

Rozdział 1

Wstęp

Obecnie wiele programów tworzonych jest w formie aplikacji przeglądarkowych. Aplikacje takie są niezależne od systemu operacyjnego użytkownika i nie wymagają instalacji, dzięki czemu są łatwiejsze i wygodniejsze w obsłudze. Przykładami takich aplikacji są produkty Google: Google Calendar oraz Google Docs, jak i ich odpowiedniki należące do firmy Microsoft, takie jak aplikacje pakietu Microsoft Office. [Dodałbym jeszcze jakieś przykłady poza GAFAM.] Pozwalają one na tworzenie różnych rodzajów plików i udostępnianie ich innym użytkownikom.

[Celem niniejszej pracy dyplomowej jest zaprojektowanie i implementacja ...] Organizer, podobnie jak wymienione aplikacje, pozwalać będzie na tworzenie różnych plików – w tym przypadku są to typy powiązane z organizacją i planowaniem: notatki, wydarzenia i zadania. Będą one mogły być grupowane w katalogach, udostępniane, dodawać będzie można powiadomienia.

Na tle innych aplikacji wyróżniać ją będzie funkcjonalność głosowania na terminy wydarzeń. Obecnie umawianie się na spotkania odbywa się na przykład na czatach internetowych, gdzie każdy musi wyrazić swoją opinię na temat preferowanej daty i godziny, przez co też często gubione są szczegóły. W aplikacji organizera, właściciel wydarzenia (użytkownik, który je utworzył) będzie mógł zdefiniować dowolną liczbę terminów, a każdy użytkownik mający do tego wydarzenia dostęp będzie mógł oddać na nie głosy. Jako że wszystko odbywa się w tym samym programie, zastosowanie tak wybranego terminu jest bardzo proste, a dodatkowe informacje dotyczące tego wydarzenia pozostają łatwo dostępne.

Podstawowymi założeniami aplikacji są intuicyjność i bezpieczeństwo. Aplikacja musi być prosta w obsłudze, aby każdy był w stanie z niej skorzystać. Użytkownicy muszą też mieć pewność, że ich hasła nie zostaną przechwycone. Projekt będzie mieć postać aplikacji webowej. Do przechowywania danych użytkowników, ich plików i innych potrzebnych informacji, jak i ich odczytu i modyfikacji, konieczne będzie wykorzystanie bazy danych – w tym przypadku MySQL. Do implementacji back-endu wykorzystany zostanie framework Spring dla języka Java, który pozwala między innymi na korzystanie z Java Persistance

API oraz ułatwia implementację autentykacji i autoryzacji użytkowników. Do wykonania front-endu posłuży framework Angular, w którym korzysta się z języka TypeScript.

Rozdział 2

Analiza tematu

2.1 Opis problemu

Aplikacja w dużym stopniu oparta jest na interakcji z użytkownikiem, który w każdym momencie użytkowania ma wiele dostępnych mu działań. Wyzwaniem jest więc nie zaprojektowanie złożonej logiki programu, a zapewnienie poprawnego działania dla wszelkich danych wejściowych pobranych od użytkownika. Zabezpieczenia takie muszą pojawić się od strony interfejsu, który powinien maksymalnie ograniczać możliwość wprowadzenia niepoprawnych danych, ale i od strony back-endu [Proszę sprawdzić, czy «back-end» jest zawsze zapisywany w ten sam sposób. Chodzi o to, żebyśmy mieli jeden konsekwentnie stosowany zapis.], gdzie weryfikować należy przychodzące zapytania HTTP. Zawartość odpowiedzi HTTP jest łatwo dostępna w inspektorze przeglądarki, dlatego już na jej poziomie nie mogą być dostępne dane wrażliwe (hasła [Czy tylko hasło, czy coś jeszcze jest daną wrażliwą?]), a nie dopiero na poziomie interfejsu.

Jedną z zalet aplikacji przeglądarkowych jest możliwość zapisywania odnośników do stron, na przykład do konkretnego pliku. Oznacza to jednak, że nawet dokładnie zaprojektowany interfejs nie pozwoli na pełną kontrolę ścieżki, którą użytkownik poruszać się będzie po programie. Zmieniając adres w przeglądarce, może on w dowolnym momencie usiłować dostać się do nieistniejących bądź nieudostępnionych mu zasobów.

Drugim problemem związanym z wykorzystaniem odnośników jest fakt, że użycie ich wiąże się z przeładowywaniem [Czy istnieje jakiś synonim? «Przeładowanie» można różnie rozumieć.] aplikacji, a co za tym idzie – z utratą przechowywanych w niej danych. Aby zachować ciągłość działania, każdorazowo konieczne są odwołania do bazy danych, a także pamięci przeglądarki: ciasteczek, pamięci lokalnej i pamięci sesji.

Każda zmiana danych wyświetlanych w interfejsie graficznym front-endu [Tak samo jak z back-endem, proszę sprawdzić, czy mamy zawsze jeden zapis.], która zachodzi w bazie danych, musi jednocześnie być niezależnie przeprowadzana na obiektach front-endu. Każdorazowe pobieranie danych z bazy skutkowałoby miganiem obrazu, a asynchronicz-

ność wykonywania operacji w back-endzie oznacza, że pobrane dane wciąż mogłyby nie odzwierciedlać stanu bazy.

2.2 Istniejące rozwiązania

2.2.1 Google Calendar

Popularną aplikacją służącą do zarządzania wydarzeniami i zadaniami jest Google Calendar. Zadania można dodać do kalendarza, ustawić ich termin oraz opis. Wydarzenia pozwalają ponadto między innymi na dodanie osób uczestniczących, miejsca wydarzenia i powiadomień.

2.2.2 Google Docs, Google Drive

Aplikacja Google Docs pozwala na tworzenie różnych rodzajów plików – dokumentów tekstowych, prezentacji multimedialnych, arkuszy kalkulacyjnych – i zapisywanie ich na dysku Google Drive. Pliki i foldery na dysku można udostępniać innym użytkownikom, dodając ich pojedynczo bądź grupowo, jak i udostępniając link. Udostępnienie ma trzy możliwe poziomy dostępu: tylko przeglądanie, komentowanie oraz edytowanie. Udostępnienie folderu przyznaje dostęp do wszystkich plików i podfolderów w nim się znajdujących.

2.2.3 Produkty Microsoft Office

Analogiczne funkcjonalności zapewniają aplikacje pakietu Microsoft Office. Programy typu OneNote, Word, PowerPoint, Excel pozwalają na tworzenie i udostępnianie plików różnych typów. Kalendarz pakietu Office pozwala na tworzenie wydarzeń, dodawanie do niego innych użytkowników i planowanie na ich podstawie spotkań w aplikacji Microsoft Teams.

[Czy istnieją jakieś rozwiązania poza GAFAM?]

Rozdział 3

Wymagania i narzędzia

3.1 Wymagania funkcjonalne

3.1.1 Użytkownicy

Znaczna większość funkcjonalności aplikacji wymaga, aby znany był aktualnie zalogowany użytkownik – od edycji profilu po przeglądanie plików. Ponadto istnieć powinien użytkownik specjalny (administrator), który zarządzać może pozostałymi użytkownikami. Wyróżnić można więc 3 rodzaje użytkowników:

U.1 User – użytkownik zwykły: edytuje profil, tworzy, przegląda, modyfikuje, usuwa, udostępnia elementy.

U.2 Admin – administrator: posiada dostęp do wszystkich funkcjonalności użytkownika zwykłego, zarządza użytkownikami – nadaje i odbiera uprawnienia administratorskie, usuwa użytkowników.

U.3 Anonymous – użytkownik niezalogowany: loguje lub rejestruje się.

Zależności między użytkownikami przedstawiono na rys. 3.1.

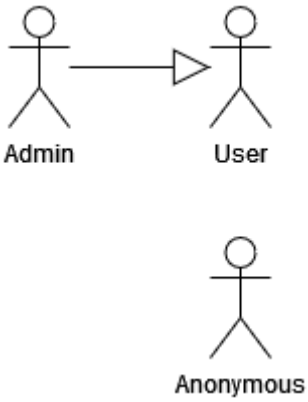
3.1.2 Funkcjonalności

Dostępne użytkownikowi funkcje aplikacji można ogólnie podzielić na operacje związane z kontem – autoryzacją i edycją – oraz związane z plikami – przeglądanie, edycja, tworzenie nowych plików.

F.1. Autentykacja użytkowników

F.1.1. Logowanie użytkownika U.3. na istniejące konto przy pomocy nazwy użytkownika oraz hasła.

F.1.2. Wylogowanie zalogowanego użytkownika (U.1., U.2.).



Rysunek 3.1: Użytkownicy aplikacji.

F.1.3. Rejestracja nowego użytkownika w systemie, umożliwiająca następnie zalogowanie (por. F.1.1.).

F.2. Zarządzanie elementami.

F.2.1. Tworzenie nowego elementu.

F.2.2. Wyświetlenie elementu. Dla należącego do innego użytkownika - zależne od przydzielonego dostępu (por. F.2.5.).

F.2.3. Edycja elementu.

F.2.3.1. Przeniesienie do innego katalogu.

F.2.3.2. Zmiana nazwy.

F.2.3.3. Mofyfikacja zawartości. Dla należącego do innego użytkownika - zależne od przydzielonego dostępu (por. F.2.5.).

F.2.3.4. Modyfikacja opcji powiadomień.

F.2.3.5. Oddawanie głosu na termin wydarzenia.

F.2.4. Usunięcie elementu.

F.2.5. Przydział dostępu innym użytkownikom do elementów.

F.3. Zarządzanie kontem użytkownika.

F.3.1. Edycja danych.

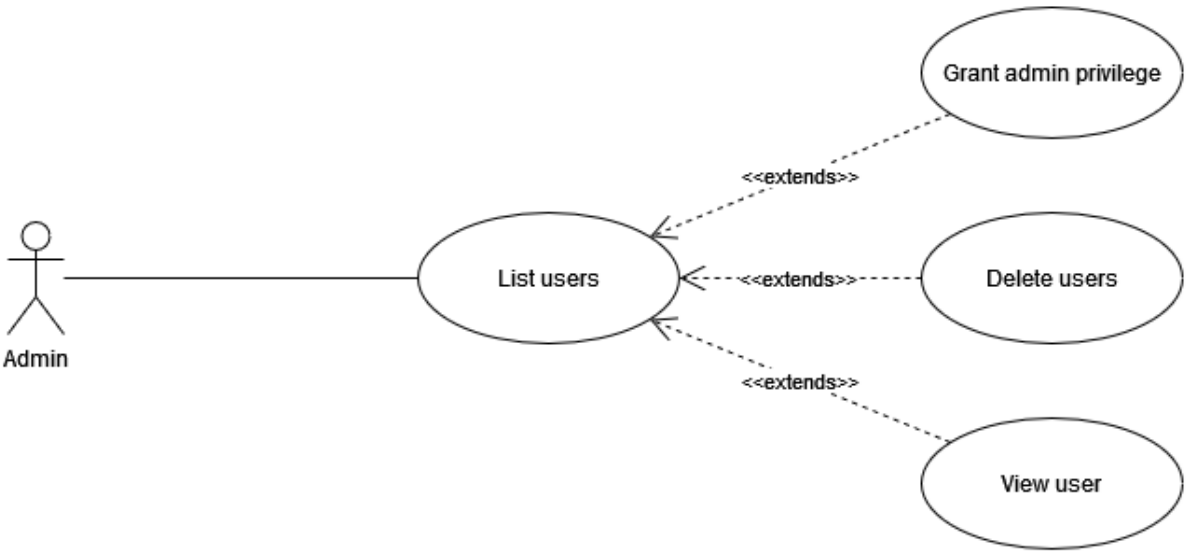
F.3.2. Usunięcie konta.

F.4. Zarządzanie użytkownikami przez administratora (U.2.).

F.4.1. Wyświetlenie wszystkich użytkowników systemu.

F.4.2. Wyświetlenie jednego użytkownika.

F.4.3. Nadanie bądź odebranie użytkownikowi uprawnień administratora.



Rysunek 3.2: Diagram przypadków użycia aplikacji dla administratora.

F.4.4. Usunięcie użytkownika.

F.5. Odporność na nieprawidłowe dane wejściowe.

Wymagania funkcjonalne dla poszczególnych użytkowników przedstawiono na rys. 3.2 [(Admin)] , [rys.] 3.3 [(User)], [rys.] 3.4 [(Anonymous)].

3.2 Wymagania niefunkcjonalne

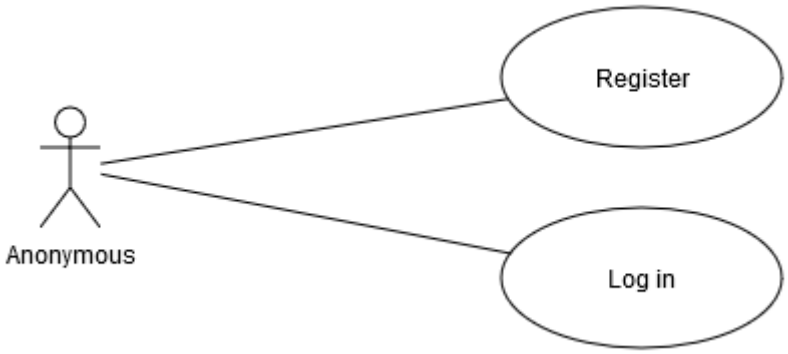
Dla aplikacji wymagającej zalogowania przez użytkownika, służącej do przechowywania prywatnych informacji niezbędne jest zapewnienie bezpieczeństwa danych przed potencjalnym przechwyceniem. Hasła użytkowników muszą być przechowywane w formie zakodowanej, do minimum należy też ograniczyć przesyłanie ich do front-endu – potrzebne są tylko do logowania, rejestracji oraz zmiany hasła.

Istotna również jest intuicyjność interfejsu użytkownika. Elementy interfejsu i ich zachowania powinny być podobne do takich, jakie można znaleźć w innych tego typu aplikacjach. Ponadto przyciski i formularze powinny zawierać informację, czego dotyczą, jakie operacje wykonują. Pomocne może być też zastosowanie ikon wraz z opisem tekstowym.

3.3 Narzędzia

Do implementacji back-endu wybrano framework Spring [14], pozwalający m.in. na pisanie aplikacji z wykorzystaniem Java Persistence API (Spring Data JPA) oraz proste zaimplementowanie autentykacji i autoryzacji użytkowników (Spring Security). Za wybraniem tego frameworku zamiast np. Entity Framework [7] przemawiała również posiadana

Rysunek 3.3: Diagram przypadków użycia aplikacji dla zwykłego, zalogowanego użytkownika.



Rysunek 3.4: Diagram przypadków użycia aplikacji dla użytkownika niezalogowanego.

już wiedza o REST API oraz chęć pogłębienia znajomości samego Springa. Wspieraną przez Spring bazę danych MySQL [10] oraz sam back-end postanowiono uruchomić w kontenerze Dockera [6].

Do testowania back-endu niezależnie od front-endu wykorzystano aplikację Postman [11]. Pozwala ona na bezpośrednie wysyłanie zapytań HTTP m.in. metodami GET, POST, PUT, DELETE z ciałem w wybranym formacie, a także na przykład na przeglądanie ciasteczek.

Do wykonania front-endu posłużył framework Angular [1, 5]. Podobnie jak w przypadku Springa, za tym wyborem przemawiało pragnienie pogłębienia wstępnej wiedzy o frameworku. Subiektywną zaletą jest też, w przeciwieństwie do popularniejszego [1] Reacta [12], wyraźny rozdział na pliki HTML, CSS (SCSS) oraz TypeScript w ramach danego komponentu. Wykorzystano również komponenty Angular Material [4], dzięki którym w prosty sposób uzyskać można estetycznie i spójnie wyglądający interfejs graficzny, zgodny z wytycznymi Material Design firmy Google [9]. Wykorzystanie gotowych komponentów pozwala na skupienie się na samej logice programu. Do operacji matematycznych na datach użyto biblioteki Luxon [8].

[1] `https://trends.stackoverflow.co/?tags=reactjs,vue.js,angular,svelte,angularjs,vuejs3`, dostęp 08.11.2024

Rozdział 4

Specyfikacja zewnętrzna

4.1 Uruchamianie

4.1.1 Wymagania

Do uruchomienia programu potrzebne są: JDK wersja 17, Docker [jaka wersja?] (w systemie Linux zainstalować można sam silnik Docker, w systemie Windows potrzebna jest aplikacja Docker Desktop) oraz Node.js, wraz z NPM (wykorzystano wersję 20). Program wykorzystywać będzie porty: 3306 (baza danych), 8080 (back-end) i 4200 (front-end) – należy więc zadbać, aby żadna inna aplikacja nie korzystała z nich podczas uruchamiania.

4.1.2 Instalacja

Jeżeli spełnione zostały powyższe wymagania, można przejść do zbudowania i uruchomienia projektu. Pliki źródłowe projektu są dostępne w repozytorium pod adresem `https://github.com/ZofiaLor/inzynier-organizer`. Należy je w wybrany sposób pobrać i, w razie potrzeby, rozpakować.

Back-end i baza danych

1. W wybranym terminalu należy ustawić bieżący katalog na `organizer`.

   Znajdować się w nim powinny pliki `mvnw` oraz `mvnw.cmd`.

2. Następnie należy uruchomić kontener zawierający bazę danych poleceniem

   `docker compose up db`

3. Aby zbudować projekt należy użyć polecenia

   - Dla systemu Linux

     `./mvnw install`

W razie niepowodzenia, należy jako root otworzyć plik

`/etc/hosts`

Dla edytora tekstowego `nano` można to zrobić następującym poleceniem:

```
sudo nano /etc/hosts
```

Należy wpisać do niego linię `127.0.0.1 db` i zapisać plik (w przypadku `nano` nacisnąć kolejno: Ctrl+O, Enter, Ctrl+X). Następnie należy powtórzyć polecenie `./mvnw install`.

- Dla systemu Windows

  ```
  .\mvnw.cmd install
  ```

  W razie niepowodzenia, należy jako administrator otworzyć plik

  `C:\Windows\System32\drivers\etc\hosts`

  Należy wpisać do niego linię `127.0.0.1 db` i zapisać plik. Następnie należy powtórzyć polecenie `.\mvnw.cmd install`.

Po poprawnym wykonaniu polecenia, wygenerowany zostanie katalog `target`, zawierający plik z rozszerzeniem `.jar`.

4. Przed uruchomieniem back-endu można zatrzymać kontener z bazą danych. Jeżeli korzysta się z programu Docker Desktop, wystarczy nacisnąć przycisk „stop” przy kontenerze „db”.

   Aby zatrzymać kontener z poziomu terminala [przecinek] należy najpierw znaleźć jego identyfikator używając polecenia

   ```
   docker ps
   ```

   Przykładowym wynikiem tego polecenia jest

   ```
   CONTAINER ID   IMAGE       COMMAND                  CREATED
   ↪  STATUS              PORTS                                NAMES
   c22ebdb94057   mysql:5.7   "docker-entrypoint.s..."   7 weeks ago
   ↪  Up About a minute   0.0.0.0:3306->3306/tcp, 33060/tcp
   ↪  organizer-db
   ```

   Następnie należy wykonać polecenie

   ```
   docker stop <CONTAINER ID>
   ```

   wstawiając w miejsce `<CONTAINER ID>` identyfikator odczytany w poprzednim kroku (tu: `c22ebdb94057`).

5. Obydwa kontenery należy zbudować poleceniem

   ```
   docker compose build
   ```

a następnie uruchomić

```
docker compose up -d
```

Front-end

1. W wybranym terminalu należy ustawić bieżący katalog na `organizer-front`.
2. Należy zainstalować wymagane pakiety poleceniami

   ```
   npm install -g @angular/cli
   ```

   oraz

   ```
   npm install
   ```

3. Aplikację należy uruchomić poleceniem

   ```
   ng serve --proxy-config profxy.config.json
   ```

4. W terminalu zostanie wypisany adres

   `http://localhost:4200/`.

   Należy otworzyć go w wybranej przeglądarce, wklejając go w polu adresu, [bez przecinka] bądź naciskając go lewym przyciskiem myszy, wciskając jednocześnie klawisz Ctrl.

Rozdział 5

Specyfikacja wewnętrzna

5.1 Implementacja klas

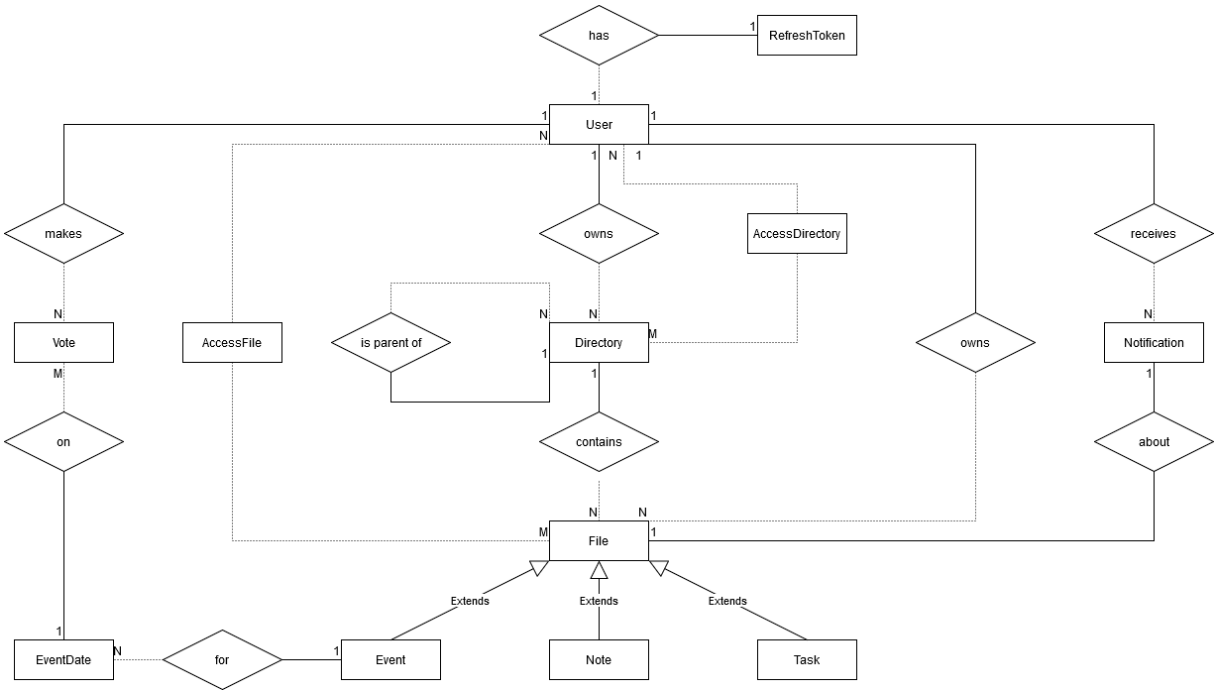
[Tutaj trzeba napisać kilka słów wprowadzenia / wstępu. Teraz zaczynamy od razu z grubej rury.]

Zależności między klasami przedstawiono na rys. 5.1.

Tabele w bazie danych mają następujące atrybuty:

- `access_directory(directory_id (PK, FK), user_id (PK, FK), access_privilege)`
- `access_file(file_id (PK, FK), user_id (PK, FK), access_privilege)`
- `directory(id (PK), name, owner_id (FK), parent_id (FK))`
- `event_date(id (PK), start, end, total_score, event_id (FK))`
- `file(id (PK), file_type, creation_date, name, text_content, start_date, end_date, location, deadline, is_finished, owner_id (FK), parent_id (FK))`
- `notification(id (PK), message, send_time_setting, is_sent, is_read, file_id (FK), user_id (FK))`
- `refresh_token(id (PK), token, expiry_date, user_id (FK))`
- `user(id (PK), username, password, name, email, role)`
- `vote(id (PK), score, user_id (FK), event_date_id (FK))`

Do zaimplementowania dziedziczenia klas `Event`, `Note` i `Task` po klasie bazowej wybrano strategię *Singe Table Inheritance*. W bazie danych wszystkie klasy reprezentowane są przez tę samą tabelę `File`, która zawiera kolumny atrybutów wszystkich dziedziczących klas, a także ukrytą kolumnę `file_type`, pozwalającą frameworkowi na rozróżnienie typów podczas mapowania obiektowo-relacyjnego. Rozwiązanie takie pozwala również na



Rysunek 5.1: Schemat ERD klas w bazie danych.

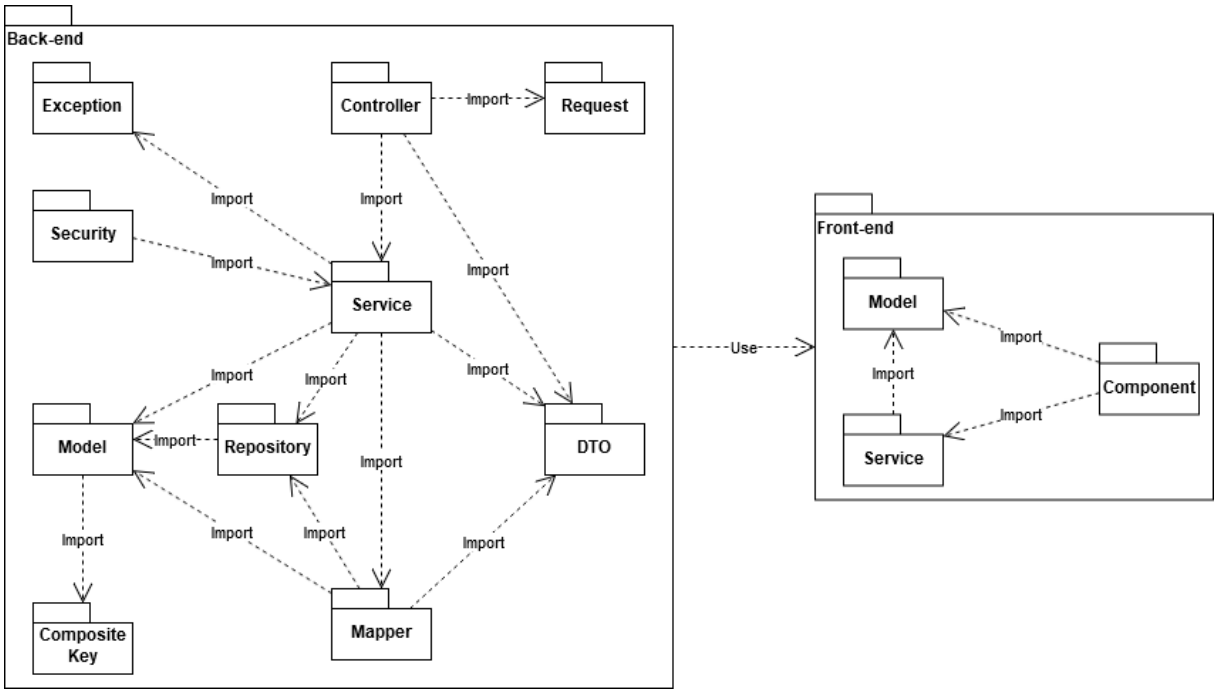
odczytanie jednocześnie wszystkich typów plików, co jest przydatne podczas wyświetlania ich w eksploratorze.

Rozwiązanie *MappedSuperclass* zostało odrzucone, ponieważ nie tworzy ono tabeli klasy bazowej, przez co nie może ona być w relacji z żadną inną klasą. Rozwiązania *Joined Table* oraz *Table Per Class* mogłyby być wykorzystane, jednak są mniej optymalne [«Optymalny» to synonim słowa «najlepszy». Mamy zatem «... jednak są mniej najlepsze ...».] z perspektywy założonego działania.

5.2 Budowa aplikacji

Na diagramie [Jaki to diagram?] [na] rys. 5.2 przedstawiono ogólną budowę wewnętrzną aplikacji. Zarówno back-end jak i front-end zostały podzielone zgodnie z konwencją wykorzystanych w nich frameworków. Framework Spring zakłada podział na klasy typu `Model`, `DTO`, `Repository`, `Service` i `Controller`, a także, opcjonalnie, `Mapper`. Angular wykorzystuje wiele komponentów, z których każdy odpowiada za część wyświetlanego interfejsu. Do komunikacji z back-endem stosuje się serwisy. W języku TypeScript istotne jest typowanie, dla klas zwracanych przez back-end zaimplementowano więc modele po stronie front-endu.

[Rys. 5.2: Jaki to diagram?]



Rysunek 5.2: Schemat UML pakietów back-endu oraz front-endu.

5.3 Autentykacja i autoryzacja

Do implementacji autentykacji i autoryzacji po stronie back-endu wykorzystano rozwiązanie inspirowane przykładem [13]. W ciasteczku przechowywany jest JWT *Json Web Token* zawierający zakodowaną nazwę użytkownika. Dla każdego zapytania wykonywany jest filtr dekodujący tę nazwę i ustawiający odpowiednie role dla kontekstu. Aby zablokować pewne metody dla użytkowników niezalogowanych, [bez przecinka] bądź niebędących administratorami, wystarczy wykorzystać adnotacje `@PreAuthorize("isAuthenticated()")` i `@PreAuthorize("hasRole('ADMIN')")`.

Ciasteczko ma ograniczony czas ważności, dlatego, wzorując się na przykładzie [15], zaimplementowano funkcjonalność odświeżania. Wykorzystano w tym celu drugie ciasteczko, o dłuższym czasie ważności. Wywołanie metody odświeżającej token przydziela nowe ciasteczka, pod warunkiem, że nie minął jeszcze czas ważności tokenu odświeżającego.

Autentykację i autoryzację po stronie front-endu wzorowano na przykładach [2] oraz [3]. Do zapisania informacji o użytkowniku wykorzystano jednak pamięć lokalną przeglądarki, nie pamięć sesji. Dzięki temu dane użytkownika są dostępne również wtedy, gdy aplikacja otwarta zostanie w nowej karcie. Pamięć sesji została za to użyta do przechowywania danych o ostatnio przeglądanych katalogach w drzewie, co sprawia, że drzewo nie wraca do stanu początkowego z każdym przeładowaniem strony. Ponadto prośba o odświeżenie ciasteczek wysyłana jest tylko wtedy, gdy endpoint [To nie jest najważniejsza rzecz na świecie. Nie będę o to kruszył kopii. Czy nie mamy tutaj pewnej niekonsekwencji ortograficznej? Mianowicie mamy «front-end» z myślnikiem, ale «endpoint» bez myślnika.]

„Send Notifications” zwróci kod błędu, jako że jest on wywoływany cyklicznie co kilka sekund i nie może zwrócić błędu, jeżeli ciasteczko zawiera poprawne dane użytkownika.

5.4 Lista endpointów

W tym podrozdziale przedstawiono listę wszystkich endpointów wystawianych przez back-end. Dla każdego opisano URL, ogólną zasadę działania, dane wejściowe oraz możliwe odpowiedzi.

5.4.1 Authorization

Log In `POST /api/auth/login`

Logowanie użytkownika.

Wymagane w ciele zapytania: nazwa użytkownika (username), hasło (password)

```
{
    "username": "newUser",
    "password": "password"
}
```

Odpowiedzi:

- Poprawne zalogowanie: obiekt użytkownika z pustym hasłem, kod 200 (*OK*)
- Niepoprawne hasło/login: „Incorrect credentials”, kod 403 (*Forbidden*)
- Brak hasła/loginu: „Empty username or password”, kod 400 (*Bad Request*)

Log Out `POST /api/auth/logout`

Wylogowanie użytkownika.

Brak wymaganego ciała zapytania.

```
{}
```

Odpowiedź: „Success”, puste ciasteczka w nagłówku, kod 200 (*OK*)

Register `POST /api/auth/register`

Rejestracja nowego użytkownika i założenie jego katalogu bazowego „Base Directory”.

Wymagana nazwa użytkownika i hasło, opcjonalne są adres email oraz imię (name). Pozostałe parametry są ignorowane.

```
{
    "username": "someUser",
    "password": "password",
    "email": "some@email.com",
    "name": "Some User"
}
```

Odpowiedzi:

- Poprawne zarejestrowanie: obiekt utworzonego użytkownika z pustym hasłem i rolą `ROLE_USER` (jeżeli jest to pierwszy użytkownik w bazie danych, rola to `ROLE_ADMIN`), kod 200 (*OK*)
- Próba zarejestrowania użytkownika o istniejącej już nazwie: kod 403 (*Forbidden*)
- Brak nazwy użytkownika lub hasła: kod 400 (*Bad Request*)

Change Password `PUT /api/auth/password`

Pozwala na zmianę hasła użytkownika.
Wymaga podania starego hasła, służącego do zatwierdzenia zmiany, oraz nowego hasła.

```
{
  "oldPassword": "pwdOld",
  "newPassword": "pwdNew"
}
```

Odpowiedzi:

- Poprawna zmiana hasła: „Success”, kod 200 (*OK*)
- Brak podanego starego lub nowego hasła: kod 400 (*Bad Request*)
- Użytkownik nie istnieje: kod 404 (*Not Found*)
- Stare hasło jest niepoprawne: kod 403 (*Forbidden*)

Grant/Revoke Admin Privilege `PUT /api/auth/grant` `PUT /api/auth/revoke`

Nadanie/odebranie roli Administratora użytkownikowi.
Wymagana nazwa użytkownika. Endpoint dostępny jest tylko administratorowi.

```
{
    "username": "newUser"
}
```

Odpowiedzi:

- Poprawne nadanie/odebranie roli: obiekt użytkownika, kod 200 (*OK*)
- Nieistniejąca nazwa użytkownika: kod 404 (*Not Found*)
- Próba nadania/odebrania sobie roli: kod 400 (*Bad Request*)
- Użytkownik nie jest administratorem: kod 403 (*Forbidden*)

Refresh Token `POST /api/refreshtoken`

Odświeża JWT odpowiadający za autoryzację użytkownika, jeżeli `refreshToken` [Chyba minted trochę się gubi w przypadku JSON-a.] zapisany w ciasteczkach nie stracił ważności.

Ciało zapytania jest ignorowane.

Odpowiedzi:

- Poprawne odświeżenie JWT: nowe ciasteczko JWT w nagłówku, kod 200 (*OK*)
- `refreshToken` stracił ważność: kod 403 (*Forbidden*)
- Nie znaleziono `refreshToken` w bazie danych: kod 404 (*Not Found*)
- `refreshToken` w ciasteczku jest pusty lub `null`: kod 400 (*Bad Request*)

5.4.2 Users

Get All Users `GET /api/users`

Zwraca wszystkich użytkowników z pustymi hasłami. Endpoint dostępny tylko administratorowi.

Odpowiedzi:

- Użytkownik jest administratorem: lista obiektów użytkowników, kod 200 (*OK*)
- Użytkownik nie jest administratorem: kod 403 (*Forbidden*)

Get All Users Safe `GET /api/users/safe`

Zwraca ID, nazwy użytkownika i nazwy wszystkich użytkowników.

Odpowiedź: lista obiektów użytkowników, kod 200 (*OK*)

Get User By ID `GET /api/users/{id}`

Zwraca użytkownika o podanym ID z pustym hasłem. Endpoint dostępny tylko administratorowi.

Odpowiedzi:

- Użytkownik istnieje: obiekt użytkownika, kod 200 (*OK*)

- Użytkownik nie istnieje: kod 404 (*Not Found*)
- Brak ID: kod 400 (*Bad Request*)
- Użytkownik nie jest administratorem: kod 403 (*Forbidden*)

Get User By ID Safe `GET /api/users/safe/{id}`

Zwraca ID, nazwę użytkownika oraz nazwę dla użytkownika o podanym ID z pustym hasłem.

Odpowiedzi:

- Użytkownik istnieje: obiekt użytkownika, kod 200 (*OK*)
- Użytkownik nie istnieje: kod 404 (*Not Found*)
- Brak ID: kod 400 (*Bad Request*)

Update User `PUT /api/users`

Aktualizuje użytkownika.

Wymagane podanie ID użytkownika w ciele zapytania. Możliwa jest tylko aktualizacja własnego użytkownika.

```
{
    "id": 5,
    "name": "New User 2"
}
```

Parametry brane pod uwagę: username, name, email.

Odpowiedzi:

- Pomyślna aktualizacja: obiekt użytkownika z pustym hasłem, kod 200 (*OK*)
- Nie znaleziono ID: kod 404 (*Not Found*)
- Nie podano ID: kod 400 (*Bad Request*)
- Próba zmiany nazwy użytkownika na już istniejącą lub pustą/zmiany użytkownika innego niż aktualnie zalogowany: kod 403 (*Forbidden*)

Delete User `DELETE /api/users/{id}`

Usuwa użytkownika o podanym ID, wraz z jego katalogami, głosami i powiadomieniami. Endpoint dostępny tylko administratorowi.

Odpowiedzi:

- ID istnieje: kod 200 (*OK*)

- ID nie istnieje: kod 404 (*Not Found*)
- Próba usunięcia własnego użytkownika/użytkownik nie jest administratorem: kod 403 (*Forbidden*)

Delete My User `DELETE /api/users/delete`

Wylogowuje i usuwa aktualnie zalogowanego użytkownika, wraz z jego katalogami, głosami i powiadomieniami.

Odpowiedzi:

- Poprawne usunięcie: puste ciasteczka w nagłówku, kod 200 (*OK*)
- Nie znaleziono użytkownika: kod 404 (*Not Found*)

5.4.3 Directories

Get My Base Directory `GET /api/directories/basedirs`

Zwraca katalog bazowy aktualnie zalogowanego użytkownika.

Odpowiedź: obiekt katalogu bazowego, kod 200 (*OK*)

Get Directories By Parent ID `GET /api/directories/subdirs/{id}`

Zwraca katalogi podrzędne katalogu o podanym ID.

Odpowiedzi:

- ID istnieje: lista obiektów katalogów, kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Nie podano ID: kod 400 (*Bad Request*)
- Brak dostępu: kod 403 (*Forbidden*)

Get Directory By ID `GET /api/directories/{id}`

Zwraca katalog o podanym ID.

Odpowiedzi:

- ID istnieje: obiekt katalogu, kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Nie podano ID: kod 400 (*Bad Request*)
- Brak dostępu: kod 403 (*Forbidden*)

Check Directory Edit Access `GET /api/directories/check/{id}`

Sprawdza, czy aktualnie zalogowany użytkownik posiada prawa do edycji katalogu o danym ID.

Odpowiedzi:

- ID istnieje: true jeżeli użytkownik może edytować katalog, false jeżeli nie, kod 200 (*OK*)
- Nie podano ID: kod 400 (*Bad Request*)
- ID nie istnieje: kod 404 (*Not Found*)

Create Directory `POST /api/directories`

Tworzy nowy katalog.

Wymagane ID rodzica, można podać nazwę. W przypadku niepodania nazwy, domyślnie ustawiana jest ona na „Unnamed Directory”.

```
{
    "name": "Test Directory",
    "parent": 3
}
```

Odpowiedzi:

- Poprawne utworzenie katalogu: obiekt nowego katalogu, kod 200 (*OK*)
- Brak ID rodzica: kod 400 (*Bad Request*)
- Nie znaleziono ID rodzica: kod 404 (*Not Found*)
- Katalog rodzic należy do innego użytkownika: kod 403 (*Forbidden*)

Update Directory `PUT /api/directories`

Aktualizuje katalog.

Wymagane podanie ID katalogu w ciele zapytania.

```
{
    "id": 2,
    "name": "Some Directory"
}
```

Parametr brany pod uwagę: name.

Odpowiedzi:

- Pomyślna aktualizacja: obiekt katalogu, kod 200 (*OK*)
- Brak obiektu/próba zmiany katalogu nadrzędnego na należący do innego użytkownika: kod 400 (*Bad Request*)
- Nie znaleziono ID: kod 404 (*Not Found*)
- Brak dostępu do edycji: kod 403 (*Forbidden*)

Delete Directory `DELETE /api/directories/{id}`

Usuwa katalog o podanym ID, wraz z jego plikami.
Odpowiedzi:

- ID istnieje: kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)

5.4.4 Files

Get Files In Directory `GET /api/files/dir/{id}`

Zwraca wszystkie pliki w katalogu o podanym ID.
Odpowiedzi:

- ID istnieje: lista obiektów katalogów, kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Brak dostępu: kod 403 (*Forbidden*)

Get File By ID `GET /api/files/{id}`

Zwraca katalog o podanym ID.
Odpowiedzi:

- ID istnieje: obiekt katalogu, kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Nie podano ID: kod 400 (*Bad Request*)
- Brak dostępu: kod 403 (*Forbidden*)

Check File Edit Access `GET /api/files/check/{id}`

Sprawdza, czy aktualnie zalogowany użytkownik posiada prawa do edycji pliku o danym ID.

Odpowiedzi:

- ID istnieje: true jeżeli użytkownik może edytować plik, false jeżeli nie, kod 200 (*OK*)
- Nie podano ID: kod 400 (*Bad Request*)
- ID nie istnieje: kod 404 (*Not Found*)

Create Event `POST /api/files/event`

Tworzy nowe wydarzenie.

Wymagane jest podanie ID katalogu rodzica. Domyślną nazwą jest „Unnamed Event".

```
{
    "name": "My First Event",
    "parent": 3,
    "textContent": "Hello World! This is my first event",
    "location": "Katowice"
}
```

Odpowiedzi:

- Pomyślne utworzenie: obiekt nowego wydarzenia, kod 200 (*OK*)
- Nie podano ID rodzica: kod 400 (*Bad Request*)
- Nie znaleziono ID rodzica: kod 404 (*Not Found*)
- Katalog rodzic należy do innego użytkownika: kod 403 (*Forbidden*)

Update Event `PUT /api/files/event`

Aktualizuje wydarzenie.

Wymagane podanie ID wydarzenia w ciele zapytania. Jeżeli daty nie zostaną podane, to zmienione zostaną na `null`.

```
{
    "id": 2,
    "location": "Gliwice"
}
```

Parametry brane pod uwagę: name, textContent, startDate, endDate, location.

Odpowiedzi:

- Pomyślna aktualizacja: obiekt wydarzenia, kod 200 (*OK*)
- Brak obiektu/próba zmiany katalogu nadrzędnego na należący do innego użytkownika: kod 400 (*Bad Request*)
- Nie znaleziono ID: kod 404 (*Not Found*)
- Brak dostępu do edycji: kod 403 (*Forbidden*)

Create Note `POST /api/files/note`

Tworzy nową notatkę.

Wymagane jest podanie ID katalogu rodzica. Domyślną nazwą jest „Unnamed Note".

```
{
  "name": "My First Note",
  "parent": 3,
  "textContent": "Hello World! This is my first note"
}
```

Odpowiedzi:

- Pomyślne utworzenie: obiekt nowej notatki, kod 200 (*OK*)
- Nie podano ID rodzica: kod 400 (*Bad Request*)
- Nie znaleziono ID rodzica: kod 404 (*Not Found*)
- Katalog rodzic należy do innego użytkownika: kod 403 (*Forbidden*)

Update Note `PUT /api/files/note`

Aktualizuje notatkę.

Wymagane podanie ID notatki w ciele zapytania.

```
{
  "id": 1,
  "name": "My Note"
}
```

Parametry brane pod uwagę: name, textContent.

Odpowiedzi:

- Pomyślna aktualizacja: obiekt notatki, kod 200 (*OK*)
- Brak obiektu/próba zmiany katalogu nadrzędnego na należący do innego użytkownika: kod 400 (*Bad Request*)
- Nie znaleziono ID: kod 404 (*Not Found*)
- Brak dostępu do edycji: kod 403 (*Forbidden*)

Create Task `POST /api/files/task`

Tworzy nowe zadanie.

Wymagane jest podanie ID katalogu rodzica. Domyślną nazwą jest „Unnamed Task”.

```
{
  "name": "My First Task",
  "parent": 2,
  "textContent": "Hello World! This is my first task",
  "isFinished": false
}
```

Odpowiedzi:

- Pomyślne utworzenie: obiekt nowego zadania, kod 200 (*OK*)
- Nie podano ID rodzica: kod 400 (*Bad Request*)
- Nie znaleziono ID rodzica: kod 404 (*Not Found*)
- Katalog rodzic należy do innego użytkownika: kod 403 (*Forbidden*)

Update Task `PUT /api/files/task`

Aktualizuje zadanie.

Wymagane podanie ID zadania w ciele zapytania. Jeżeli data wykonania nie zostanie podana, to zmieniona zostanie na `null`.

```
{
  "id": 3,
  "isFinished": true
}
```

Parametry brane pod uwagę: name, textContent, isFinished, deadline [← w minted].

Odpowiedzi:

- Pomyślna aktualizacja: obiekt zadania, kod 200 (*OK*)
- Brak obiektu/próba zmiany katalogu nadrzędnego na należący do innego użytkownika: kod 400 (*Bad Request*)
- Nie znaleziono ID: kod 404 (*Not Found*)
- Brak dostępu do edycji: kod 403 (*Forbidden*)

Delete File `DELETE /api/files/{id}`

Usuwa plik o podanym ID, w przypadku wydarzenia - wraz z jego obiektami EventDate.

Odpowiedzi:

- ID istnieje: kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)

5.4.5 Access Directory

Get AccessDirectory By User `GET /api/ad/user/{user}`

Zwraca listę obiektów `AccessDirectory` dla podanego ID użytkownika.

Lista jest pusta dla nieistniejącego użytkownika.

Odpowiedzi:

- ID istnieje: lista obiektów `AccessDirectory`, kod 200 (*OK*)
- Brak ID: kod 400 (*Bad Request*)

Get AccessDirectory By Directory `GET /api/ad/dir/{dir}`

Zwraca listę obiektów `AccessDirectory` dla podanego ID katalogu.

Lista jest pusta dla nieistniejącego katalogu.

Odpowiedzi:

- ID istnieje: lista obiektów `AccessDirectory`, kod 200 (*OK*)
- Brak ID: kod 400 (*Bad Request*)

Modify AccessDirectory `POST /api/ad`

Tworzy lub aktualizuje obiekt `AccessDirectory` o podanych ID użytkownika i katalogu.

Wymagane podanie obydwu ID w ciele zapytania.

```
{
    "id": {
        "userId": 4,
        "directoryId": 3
    },
    "accessPrivilege": 1
}
```

Odpowiedzi:

- Pomyślne utworzenie/aktualizacja: obiekt `AccessDirectory`, kod 200 (*OK*)
- Nie istnieje któreś z ID: kod 404 (*Not Found*)
- Brak ID: kod 400 (*Bad Request*)

Delete AccessDirectory `DELETE /api/ad/{user}/{dir}`

Usuwa obiekt `AccessDirectory` o podanych ID użytkownika i katalogu.
Odpowiedzi:

- ID istnieją: kod 200 (*OK*)
- ID nie istnieją: kod 404 (*Not Found*)

5.4.6 Access File

Get AccessFile By User `GET /api/af/user/{user}`

Zwraca listę obiektów `AccessFile` dla podanego ID użytkownika.
Lista jest pusta dla nieistniejącego użytkownika.
Odpowiedzi:

- ID istnieje: lista obiektów `AccessFile`, kod 200 (*OK*)
- Brak ID: kod 400 (*Bad Request*)

Get AccessFile By File `GET /api/af/file/{file}`

Zwraca listę obiektów `AccessFile` dla podanego ID pliku.
Lista jest pusta dla nieistniejącego pliku.
Odpowiedzi:

- ID istnieje: lista obiektów `AccessFile`, kod 200 (*OK*)
- Brak ID: kod 400 (*Bad Request*)

Modify AccessFile `POST /api/af`

Tworzy lub aktualizuje obiekt `AccessFile` o podanych ID użytkownika i pliku.
Wymagane podanie obydwu ID w ciele zapytania.

```
{
    "id": {
        "userId": 4,
        "fileId": 3
    },
    "accessPrivilege": 1
}
```

Odpowiedzi:

- Pomyślne utworzenie/aktualizacja: obiekt `AccessFile`, kod 200 (*OK*)
- Nie istnieje któreś z ID: kod 404 (*Not Found*)
- Brak ID: kod 400 (*Bad Request*)

Delete AccessFile `DELETE /api/af/{user}/{file}`

Usuwa obiekt `AccessFile` o podanych ID użytkownika i pliku.
Odpowiedzi:

- ID istnieją: kod 200 (*OK*)
- ID nie istnieją: kod 404 (*Not Found*)

5.4.7 Event Dates

Get All EventDates `GET /api/ed`

Zwraca wszystkie obiekty `EventDate`.
Odpowiedź: lista obiektów `EventDate`, kod 200 (*OK*)

Get EventDates By Event ID `GET /api/ed?id={eventId}`

Zwraca obiekty `EventDate` dotyczące wydarzenia o podanym ID.
Odpowiedzi:

- ID istnieje: lista obiektów `EventDate`, kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Nie podano ID: kod 400 (*Bad Request*)

Get EventDate By ID `GET /api/ed/{id}`

Zwraca obiekt `EventDate` o podanym ID.
Odpowiedzi:

- ID istnieje: obiekt `EventDate`, kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Nie podano ID: kod 400 (*Bad Request*)

Create EventDate `POST /api/ed`

Tworzy obiekt `EventDate`.

Wymagane jest podanie ID wydarzenia, a także początku i końca terminu. Wynik całkowity ustawiany jest na 0. Wydarzenie musi istnieć i należeć do tworzącego obiekt użytkownika.

```
{
    "event": 2,
    "start" : "2024-10-18T12:00:00",
    "end": "2024-10-18T12:30:00"
}
```

Odpowiedzi:

- Pomyślne utworzenie: nowy obiekt `EventDate`, kod 200 (*OK*)
- Brak ID/terminu początkowego/końcowego: kod 400 (*Bad Request*)
- Wydarzenie nie istnieje: kod 404 (*Not Found*)
- Wydarzenie należy do innego użytkownika: kod 403 (*Forbidden*)

Delete EventDate `DELETE /api/ed/{id}`

Usuwa obiekt `EventDate` o podanym ID, wraz z jego głosami. Wydarzenie należeć musi do usuwającego obiekt użytkownika.

Odpowiedzi:

- ID istnieje: kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Wydarzenie należy do innego użytkownika: kod 403 (*Forbidden*)

5.4.8 Votes

Get Votes By EventDate ID `GET /api/votes/ed/{id}`

Zwraca głosy na termin EventDate o podanym ID.

Odpowiedzi:

- ID istnieje: lista obiektów głosów, kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Nie podano ID: kod 400 (*Bad Request*)

Get Current User's Vote By EventDate ID `GET /api/votes/myvote/{id}`

Zwraca listę głosów oddanych przez aktualnie zalogowanego użytkownika na termin `EventDate` o podanym ID.

Odpowiedzi:

- ID istnieje: lista obiektów głosów, kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Nie podano ID: kod 400 (*Bad Request*)

Cast Vote `POST /api/votes`

Sprawdza, czy zalogowany użytkownik oddał głos na dany termin, jeżeli nie, to tworzy nowy głos, jeżeli tak, to aktualizuje istniejący. Modyfikuje wynik całkowity dla terminu `EventDate`. Użytkownik musi posiadać dostęp do wydarzenia, na które głosuje.

Wymagane podanie ID `EventDate`.

```
{
    "eventDate": 1,
    "score": 1
}
```

Odpowiedzi:

- Pomyślne oddanie/modyfikacja głosu: obiekt głosu, kod 200 (*OK*)
- Nie podano ID `EventDate`: kod 400 (*Bad Request*)
- Nie istnieje ID `EventDate`: kod 404 (*Not Found*)
- Brak dostępu do wydarzenia: kod 403 (*Forbidden*)

Delete Vote `DELETE /api/vote/{id}`

Usuwa głos o podanym ID. Głos musiał być oddany przez aktualnie zalogowanego użytkownika.

Odpowiedzi:

- ID istnieje: kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Głos należy do innego użytkownika: kod 403 (*Forbidden*)

5.4.9 Notifications

Get All My Notifications `GET /api/notifs/mynotifs`

Zwraca wszystkie wysłane powiadomienia aktualnie zalogowanego użytkownika.

Odpowiedź: lista obiektów powiadomień, kod 200 (*OK*)

Get All My Read/Unread Notifications `GET /api/notifs/mynotifs?read={read}`

Zwraca wszystkie wysłane odczytane/nieodczytane powiadomienia aktualnie zalogowanego użytkownika, w zależności od parametru `read` (**`true`** - [–] odczytane, **`false`** - nieodczytane).

Odpowiedź: lista obiektów powiadomień, kod 200 (*OK*)

Get Current User's Notification By File ID `GET /api/notifs/file/{id}`

Zwraca listę niewysłanych powiadomień aktualnie zalogowanego użytkownika powiązanych z plikiem o podanym ID.

Odpowiedzi:

- ID istnieje: lista obiektów powiadomień, kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Nie podano ID: kod 400 (*Bad Request*)

Create Notification `POST /api/notifs`

Tworzy powiadomienie dla aktualnie zalogowanego użytkownika. Użytkownik musi mieć dostęp do pliku.

Wymagane jest podanie ID pliku. Czas wysłania domyślnie ustawiany jest na czas utworzenia, powiadomienie jest domyślnie nieodczytane. Wysłanie odbywa się na podstawie porównania aktualnego czasu z czasem wysłania.

```
{
  "file": 3,
  "message": "Test notif",
  "sendTimeSetting": "2024-10-28T15:30:00"
}
```

Odpowiedzi:

- Pomyślne utworzenie: nowy obiekt powiadomienia, kod 200 (*OK*)
- Brak ID pliku: kod 400 (*Bad Request*)
- Nie istnieje ID pliku: kod 404 (*Not Found*)
- Brak dostępu do pliku: kod 403 (*Forbidden*)

Update Notification `PUT /api/notifs`

Akutalizuje powiadomienie o podanym ID. Musi ono należeć do aktualnie zalogowanego użytkownika

Wymagane jest podanie ID powiadomienia. Jedynym branym pod uwagę parametrem jest **read**.

```
{
  "id": 1,
  "read": true
}
```

Odpowiedzi:

- Pomyślne utworzenie: nowy obiekt powiadomienia, kod 200 (*OK*)
- Brak ID powiadomienia: kod 400 (*Bad Request*)
- Nie istnieje ID powiadomienia: kod 404 (*Not Found*)
- Powiadomienie należy do innego użytkownika: kod 403 (*Forbidden*)

Send Current User's Notifications `PUT /api/notifs/send`

Wysyła powiadomienia aktualnie zalogowanego poprzez porównanie ich czasu wysłania z czasem aktualnym.

Odpowiedzi:

- Poprawna aktualizacja powiadomień: kod 200 (*OK*)
- Nie znaleziono użytkownika: kod 404 (*Not Found*)

Delete Notification `DELETE /api/notifs/{id}`

Usuwa powiadomienie o podanym ID. Musi ono należeć do aktualnie zalogowanego użytkownika.

Odpowiedzi:

- ID istnieje: kod 200 (*OK*)
- ID nie istnieje: kod 404 (*Not Found*)
- Powiadomienie należy do innego użytkownika: kod 403 (*Forbidden*)

Rozdział 6

Weryfikacja i walidacja

Testowanie aplikacji wymaga sprawdzenia, czy jest ona odporna na nieprawidłowe dane wejściowe oraz czy dla danych poprawnych zachowuje się zgodnie z oczekiwaniami. Przetestować pod tym kątem należy zarówno back-end, jak i front-end.

W przypadku back-endu należy zweryfikować, czy faktyczne odpowiedzi są zgodne z wymienionymi w podrozdziale Lista endpointów. Z kolei front-end powinien już na poziomie interfejsu w jak największym stopniu ograniczyć możliwość wprowadzenia błędnych danych.

6.1 Testowanie back-endu

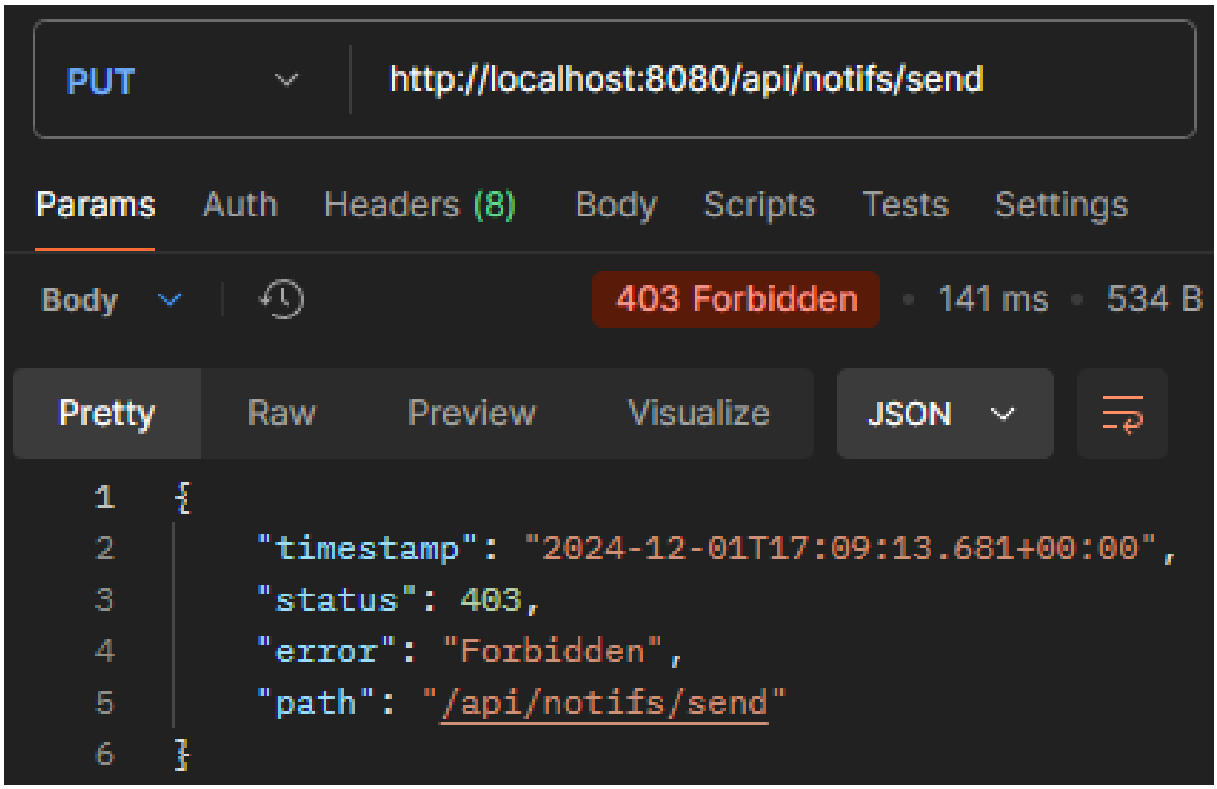
Testowanie rozpoczęto od sprawdzenia przypadków użycia dla użytkownika niezalogowanego. Większość endpointów wymaga autentykacji, sprawdzono więc, że zwracają one bez niej kod 403 (*Forbidden*). Przykład takiego zapytania pokazano na rys. 6.1.

Przy rejestracji, zgodnie z założeniami, otrzymano kod 400 gdy w ciele zapytania brakowało hasła lub nazwy użytkownika, oraz kod 403 gdy wpisano nazwę użytkownika występującą już w bazie danych. Dla poprawnych danych:

```
{
        "username": "test",
        "password": "password",
        "name": "Test User"
}
```

Otrzymano odpowiedź:

```
{
        "id": 22,
        "username": "test",
        "name": "Test User",
```



Rysunek 6.1: Zrzut ekranu z programu Postman przedstawiający próbę użycia endpointu, do którego użytkownik nie posiada dostępu.

```
        "email": null,
        "password": "",
        "role": "ROLE_USER",
        "directories": null,
        "votes": null,
        "notifications": null
}
```

Odpowiedź nie zawiera już hasła – zostało ono zakodowane i zapisane w bazie.

Rejestrację przeprowadzono również przy pustej bazie danych. Pierwszy utworzony użytkownik otrzymał rolę `ROLE_ADMIN`, każdy kolejny – `ROLE_USER`.

Podobnie przetestowano logowanie: dla brakujących danych otrzymano kod 400 i informację „Empty username or password”, dla nieistniejącego loginu i nieprawidłowego hasła – kod 403 i informację „Incorrect credentials”. Dla poprawnych danych logowania:

```
{
        "username": "test",
        "password": "password"
}
```

Otrzymano odpowiedź:

```
{
        "id": 22,
        "username": "test",
        "name": "Test User",
        "email": null,
        "password": "",
        "role": "ROLE_USER",
        "directories": [
                26
        ],
        "votes": [],
        "notifications": []
}
```


Obiekt użytkownika został pobrany z bazy danych. Automatycznie utworzony został katalog bazowy, który jest jedynym katalogiem użytkownika nieposiadającym katalogu nadrzędnego.

Ponadto wygenerowane zostały dwa ciasteczka:


```
organizerJwt=eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXN0IiwiaWF0IjoxNzMzMDc1MTYxLCJleHAiOjE3
↪ Path=/; HttpOnly; Expires=Mon, 02 Dec 2024 17:46:00 GMT;
organizerRefreshJwt=1ee7c994-e296-4a12-a636-8cf39380a1e8;
↪ Path=/api/auth/refreshtoken; HttpOnly; Expires=Mon, 02 Dec 2024
↪ 23:46:00 GMT;
```


Następnie przetestowano przypadki użycia dla użytkownika zwykłego.

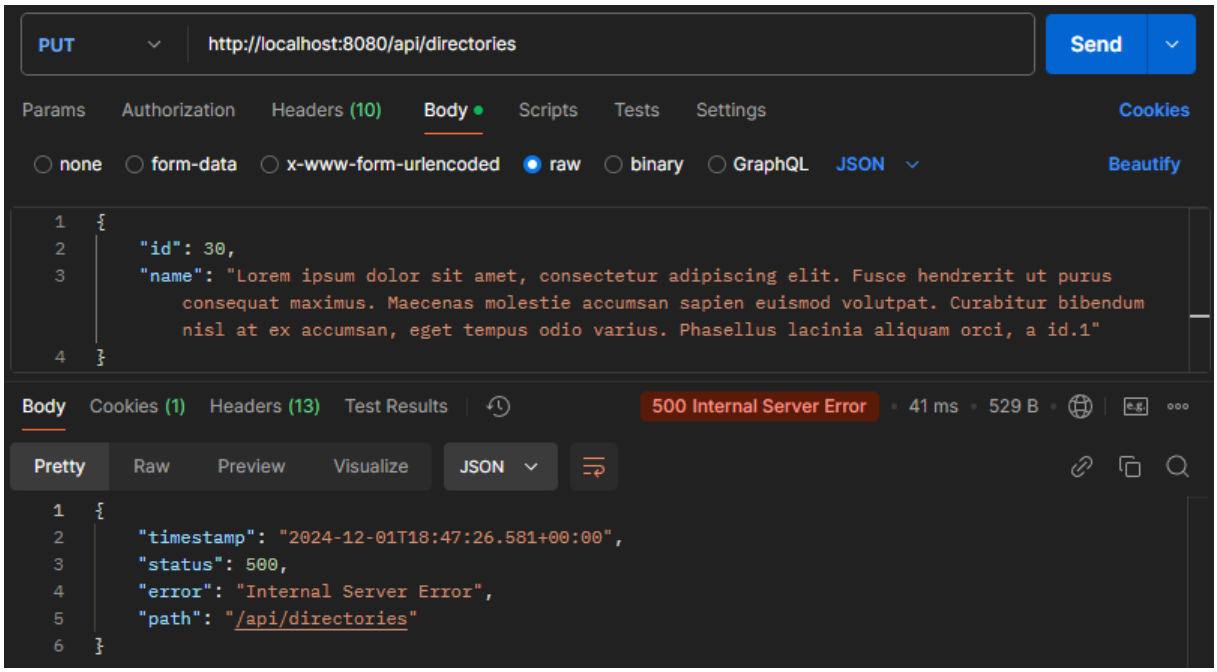
Wylogowanie przebiega zawsze pomyślnie, można je powtarzać nawet będąc już wylogowanym bez negatywnych skutków. W wyniku wylogowania obydwa ciasteczka stają się puste.

Przetestowano działanie odświeżania ciasteczka organizerJwt. Przed odświeżeniem ciasteczka miały następujące wartości:


```
organizerJwt=eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJuZXdVc2VyIiwiaWF0IjoxNzMzMTY2NDA4LCJleHAi
↪ Path=/; HttpOnly; Expires=Tue, 03 Dec 2024 19:06:48 GMT;
organizerRefreshJwt=8d9fab79-5a99-4bb0-b56b-ad0626fb2d61;
↪ Path=/api/auth/refreshtoken; HttpOnly; Expires=Wed, 04 Dec 2024
↪ 01:06:48 GMT;
```


Po wykonaniu odświeżenia:


```
organizerJwt=eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJuZXdVc2VyIiwiaWF0IjoxNzMzMTY2NDg0LCJleHAi
↪ Path=/; HttpOnly; Expires=Tue, 03 Dec 2024 19:08:04 GMT;
```



Rysunek 6.2: Zrzut ekranu z programu Postman przedstawiający nieudaną próbę zmiany nazwy katalogu na tekst o długości 256 znaków.

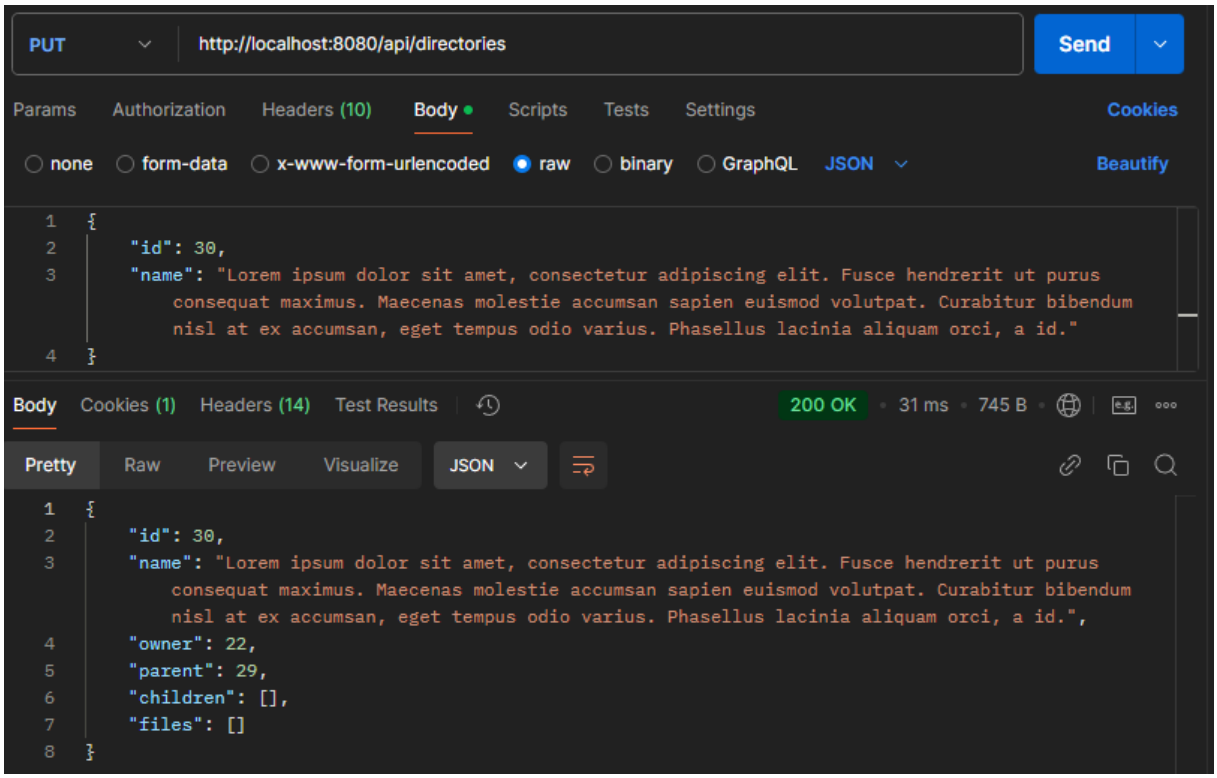
```
organizerRefreshJwt=8d9fab79-5a99-4bb0-b56b-ad0626fb2d61;
↪ Path=/api/auth/refreshtoken; HttpOnly; Expires=Wed, 04 Dec 2024
↪ 01:08:04 GMT;
```

Próba dostępu do endpointu dostępnego tylko administratorowi, na przykład nadania roli administratora, skutkuje kodem 403 odpowiedzi.

Przetestowano tworzenie nowych elementów. Kody błędów 400, 403 i 404 są zwracane zgodnie z oczekiwaniami. Elementy tworzone są poprawnie, można je tworzyć tylko we własnych katalogach – w przeciwnym razie zwracany jest kod 403. Kolumna textContent jest typu `TEXT` – jest to typ danych bazy MySQL o rozmiarze 64KB. Pozostałe kolumny tekstowe posiadają domyślny typ `TINYTEXT`, mogą więc mieć maksymalnie 255 znaków. Próba przypisania większej liczby znaków skutkuje wyjątkiem w bazie danych. Przykład takiej sytuacji pokazano na rys. 6.2 oraz 6.3 Zabezpieczenie przed nim występuje po stronie front-endu. Back-end nie weryfikuje też, czy termin początkowy wydarzenia jest wcześniejszy od końcowego.

Z takim samym wynikiem przeprowadzono testy edycji nazw i zawartości elementów. Notatki, wydarzenia oraz zadania posiadają osobne endpointy do ich edycji. Próba edycji elementu przy użyciu niewłaściwego endpointu, na przykład wydarzenia używając `PUT /api/files/note`, skutkuje kodem 404.

Element można przenieść tylko do istniejącego katalogu, należącego do tego samego użytkownika, przez tego użytkownika. Ignorowane są przypadki, gdy podejmowana jest próba przeniesienia katalogu bazowego, bądź ustawienia katalogu nadrzędnego na ten sam katalog lub katalog podrzędny. Taka sytuacja mogłaby skutkować nieskończoną pętlą



Rysunek 6.3: Zrzut ekranu z programu Postman przedstawiający udaną próbę zmiany nazwy katalogu na tekst o długości 255 znaków.

podczas sprawdzania dostępu, a także błędem bazy danych przy usuwaniu.

```
{
        "id": 29,
        "parent": 29
}
```

Odpowiedź:

```
{
        "id": 29,
        "name": "Child Directory",
        "owner": 22,
        "parent": 26,
        "children": [
                30
        ],
        "files": []
}
```

Przetestowano odczytywanie elementów, odpowiedzi były zgodne z wymienionymi na liście endpointów.

W katalogu o ID 29 umieszczono katalog podrzędny oraz plik:


```
{
        "id": 29,
        "name": "Child Directory",
        "owner": 22,
        "parent": 26,
        "children": [
                30
        ],
        "files": [
                15
        ]
}
```


Następnie usunięto katalog 29. Próba odczytu katalogu 29, 30 oraz pliku 15 skutkuje błędem 404 – zawartość katalogu została kaskadowo usunięta.

W przypadku katalogu bazowego, próba usunięcia skutkuje kodem błędu 403.

Sprawdzanie uprawnień do edycji zwraca kod 404 dla nieistniejącego elementu, `true` lub `false` odpowiednio kiedy użytkownik posiada lub nie posiada uprawnień do edycji elementu.

Przetestowano działanie algorytmu dostępu dla użytkownika o ID 4. Zdefiniowano dla niego dostęp do katalogu 3 z poziomem dostępu 1 (odczyt) oraz do plików 1 i 2 z poziomem dostępu 2 (edycja). Struktura wymienionych plików:

```
katalog 1
        katalog 2
                katalog 11
                        plik 2
        katalog 3
                plik 1
                plik 3
```

Próba odczytu plików 1, 2 i 3 oraz katalogu 3 przebiega pomyślnie. Wysłanie zapytania GET dla katalogów 1, 2 i 11 skutkuje kodem 403. Pomyślnie przebiega edycja plików 1 i 2, natomiast dla pliku 3 otrzymywany jest kod 403.

12 grudnia 2024 roku o godzinie 18:10 dla użytkownika 4 dodano powiadomienie do pliku 1.


```
{
        "id": 6,
        "user": 4,
```

```
        "file": 1,
        "message": "Test notif",
        "sendTimeSetting": "2024-12-02T18:15:00",
        "read": false,
        "sent": false
}
```


Wysłano od razu zapytanie Send Notifications. Zapytanie Get All Notifications dla użytkownika zwraca pustą listę, a zapytanie Get All Unsent Notifications dla pliku zwraca listę zawierającą to powiadomienie. Po godzinie 18:15 powtórzono wysłanie powiadomień i ponownie wykonano dwa zapytania Get. Tym razem powiadomienie zwrócone zostało przez endpoint `GET /api/notifs/mynotifs` zwróciło listę z powiadomieniem, którego wartość pola `sent` została zmieniona na `true`. Lista niewysłanych powiadomień jest pusta.

Utworzono powiadomienie z wcześniejszym terminem wysłania.


```
{
        "id": 7,
        "user": 4,
        "file": 1,
        "message": "Test notif 2",
        "sendTimeSetting": "2024-12-02T17:15:00",
        "read": false,
        "sent": true
}
```


Powiadomienie od początku jest wysłane.

Powiadomienie można dodać tylko do pliku, do którego posiada się dostęp. Zostaje ono przypisane zalogowanemu użytkownikowi.

Powiadomienie zaktualizowano, zmieniając wartość pola `read` na „true” – jest to jedyne pole, które można zmienić. Usunięto własne powiadomienie, następnie spróbowano zaktualizować i usunąć powiadomienie innego użytkownika – otrzymano kod 403.

Próba utworzenia obiektu `EventDate` dla wydarzenia należącego do innego użytkownika skutkuje kodem 403, dla nieistniejącego wydarzenia – kodem 404. Podanie ID pliku innego niż wydarzenie również zwraca kod 404. Próba utworzenia wydarzenia z pustym terminem początkowym zwraca kod 400, można jednak nie podać terminu końcowego:


```
{
        "id": 7,
        "event": 16,
        "votes": null,
        "totalScore": 0,
```

```
        "start": "2024-11-18T12:00:00",
        "end": null
}
```


Oddawanie wielu głosów na ten sam termin `EventDate` modyfikuje wartość `score`, nie dodaje kolejnych głosów. Zalogowano się jako inny użytkownik i oddano głos: wartość `totalScore` dla terminu jest każdorazowo przeliczana, jest równa sumie punktów z każdego głosu. Głos można oddać tylko na wydarzenie, do którego posiada się dostęp – w przeciwnym razie zwracany jest kod 403.

Zwykły użytkownik nie może listować pełnych danych innych użytkowników (kod 403), może jednak skorzystać z bezpiecznego endpointu – dla wszystkich użytkowników oraz użytkownika o konkretnym ID. Fragment odpowiedzi dla bezpiecznego listowania wszystkich użytkowników:


```
{
        "id": 4,
        "username": "newUser",
        "name": "User",
        "email": "",
        "password": null,
        "role": null,
        "directories": null,
        "votes": null,
        "notifications": null
},
{
        "id": 5,
        "username": "newUser2",
        "name": "New User 2",
        "email": "",
        "password": null,
        "role": null,
        "directories": null,
        "votes": null,
        "notifications": null
},
```


Próba edycji danych innego użytkownika niż zalogowany skutkuje kodem błędu 403. Nie można też zmienić nazwy na inną występującą już w bazie. Wyjątkiem jest podanie własnej nazwy użytkownika. Podanie pustego ciągu znaków jako nowa nazwa użytkownika

zwraca kod 403. Zmiany hasła i roli są ignorowane. Na puste ciągi znaków można zmieniać pola `name` oraz `email`.

Przetestowano zmianę hasła. Jeżeli nie podane zostanie stare lub nowe hasło otrzymywany jest kod 400. Dla niepoprawnego starego hasła zwracany jest kod 403. Zmieniono hasło w poprawny sposób i spróbowano zalogować się przy pomocy starego i nowego hasła – operacja przebiegła pomyślnie dla hasła nowego.

Zwykły użytkownik może usunąć tylko swoje konto. Usunięto użytkownika:


```
{
        "id": 22,
        "username": "test",
        "name": "Test User",
        "email": null,
        "password": "",
        "role": "ROLE_USER",
        "directories": [
                26
        ],
        "votes": [
                13
        ],
        "notifications": []
}
```


Automatycznie nastąpiło wylogowanie – endpointy dostępne każdemu zalogowanemu użytkownikowi zwracają kod 403, ciasteczka organizerJwt i organizerRefreshJwt są puste. Zalogowano się na konto innego użytkownika. Próby odczytu katalogu o ID 26 i głosów dla terminów wydarzeń użytkownika zwracają kod 404.

Zalogowano się jako administrator. Możliwe było wylistowanie pełnych danych użytkowników (nie licząc haseł). Fragment odpowiedzi:


```
{
        "id": 4,
        "username": "newUser",
        "name": "User",
        "email": "newuser@email.com",
        "password": "",
        "role": "ROLE_ADMIN",
        "directories": [
                4,
                12,
```

```
                13
        ],
        "votes": [
                1,
                7,
                9,
                11
        ],
        "notifications": [
                6,
                7,
                8
        ]
},
{
        "id": 5,
        "username": "newUser2",
        "name": "New User 2",
        "email": "newuser2@email.com",
        "password": "",
        "role": "ROLE_USER",
        "directories": [
                5
        ],
        "votes": [],
        "notifications": []
},
```


Nadano i odebrano innym użytkownikom rolę administratora. Próba zmiany własnej roli zwróciła kod 400.

Próba usunięcia własnego użytkownika poprzez endpoint administratora zwraca kod 403. Usunięcie innego użytkownika przebiegło pomyślnie.

6.2 Testowanie front-endu

Podczas testowania front-endu, tam, gdzie nie zaznaczono inaczej, korzystano z przeglądarki Mozilla Firefox w wersji 133.0. Wysyłane do back-endu zapytania weryfikowano przy pomocy inspektora przeglądarki.

Po przejściu na adres `http://localhost:4200/` w przeglądarce, pokazana zostaje

strona powitalna. Użytkownik jest niezalogowany – w górnym pasku nawigacyjnym dostępny jest przycisk „Log In”.

Sprawdzono zabezpieczenia przed próbą dostępu do nieudostępnionych użytkownikowi niezalogowanemu zasobów. Przejście na stronę `http://localhost:4200/file/1` przeładowuje stronę, nie zostaje jednak wysłane żadne zapytanie, a widok strony nie zmienia się. Próba wejścia na stronę profilu `http://localhost:4200/profile` skutkuje wyświetleniem komunikatu „Please log in to view your profile”. Próba wejścia na dostępną administratorowi stronę `http://localhost:4200/admin/users` pokazuje komunikat „This page is accessible for administrators only”. Przejście na niezdefiniowany adres, na przykład `http://localhost:4200/wrong` przekierowuje na prostą stronę „Page Not Found”, zawierającą odnośnik do strony głównej.

Na stronie rejestracji (`http://localhost:4200/auth/register`) sprawdzono, że nie można się zarejestrować bez podania nazwy użytkownika oraz przynajmniej 6-znakowego hasła – przycisk zatwierdzający jest nieaktywny. Pole adresu e-mail może pozostać puste, jeśli jednak wpisze się do niego treść, to musi ona być poprawnym adresem. Wpisano nazwę użytkownika występującą już w bazie danych: po zatwierdzeniu na dole ekranu pojawiło się powiadomienie „This username is already taken”. Wpisano unikalną nazwę użytkownika, po zatwierdzeniu na dole ekranu pojawiło się powiadomienie „You have successfully registered!”, wyświetlił się również link do strony logowania.

Na stronie logowania (`http://localhost:4200/auth/login`) również nie można się zalogować bez podania nazwy użytkownika i hasła. Podanie nieprawidłowej kombinacji nazwy i hasła, w tym niewystępującej w bazie danych nazwy użytkownika, skutkuje pojawieniem się powiadomienia „Incorrect username or password”. Po wpisaniu poprawnych danych użytkownik zostaje zalogowany i przekierowany na stronę główną, która zawiera przeglądarkę plików. Dla nowo zarejestrowanego użytkownika na liście plików znajduje się jeden katalog – „Base Directory” – utworzony automatycznie podczas rejestracji.

Aby zweryfikować automatyczne odświeżanie ciasteczek, tymczasowo zmieniono ich czas ważności w back-endzie na 1 minutę dla `organizerJwt` i 3 minuty dla `organizerRefreshJwt`. W inspektorze przeglądarki sprawdzono, że co jedną minutę cyklicznie wywoływane zapytanie wysyłające powiadomienia zwraca kod 403 i wysyłana jest prośba o odświeżenie ciasteczka. Następnie zamknięto kartę aplikacji i ponownie otwarto ją po ponad 3 minutach. Użytkownik został automatycznie wylogowany.

Utworzono dwa nowe katalogi oraz notatkę, wydarzenie i zadanie. Formularz nie pozwala na wpisanie nazwy elementu lub lokalizacji dłuższej niż 50 znaków, ani zawartości pliku dłuższej niż 65000 znaków, nie pozwala też zapisać elementu z pustą nazwą – przycisk do zapisu jest nieaktywny. Sprawdzono, że poprawnie zapisuje się notatka z 65000-znakową treścią. Nie można zapisać wydarzenia, którego termin początkowy jest późniejszy od końcowego. Jeżeli w polu daty wpisze się niepełną datę (na przykład bez minut), lub będzie ona nieprawidłowa (na przykład 30.02.2024) pojawi się informacja,

że data ta nie zostanie zapisana w bazie danych. Próba podania daty spoza przedziału 1970-01-01 00:00:01 UTC do 2038-01-19 03:14:07 UTC skutkuje informacją, że data nie mieści się w tym przedziale. Przycisk „Discard changes" przywraca stan formularza do stanu ostatniego zapisu.

Struktura utworzonych w tym kroku plików:

```
Base Directory
        Child Directory
                Grandchild Directory
                        New Event
                        New Task
                New Note
```

W panelu udostępniania elementów wybrano z listy użytkowników `newUser` i udzielono mu dostępu do edycji katalogu „Child Directory" oraz dostępu do przeglądania pliku „New Task". Zalogowano się jako `newUser`. Zweryfikowano, że użytkownik ten może edytować pliki „New Note" oraz „New Event", a tylko przeglądać „New Task".

Następnie, jako właściciel, usunięto udzielenie dostępu do przeglądania pliku „New Task". Ponownie zalogowano się jako `newUser` – tym razem może on edytować ten plik. Zmodyfikowano dostęp do katalogu „Child Directory" na dostęp do przeglądania. Zgodnie z założeniami, użytkownik newUser może tylko przeglądać wszystkie pliki w tym katalogu.

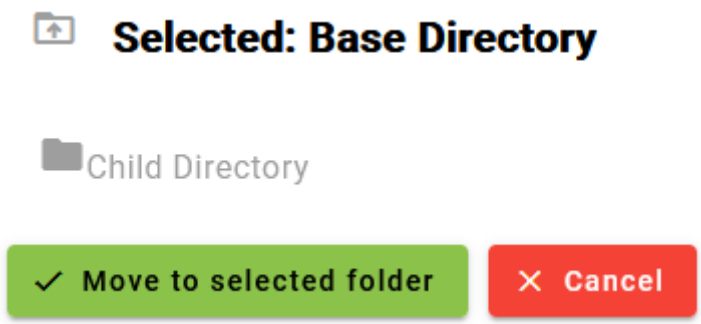
Panel udostępniania pobiera listę użytkowników z bazy danych, nie jest możliwe wybranie nieistniejącego użytkownika.

Pliki i katalogi nieudostępnione użytkownikowi nie pojawiają się w drzewie plików. Jeżeli użytkownik spróbuje dostać się do elementu wpisując jego identyfikator w adresie URL, w przeglądarce nie pojawi się żaden element, a na dole ekranu pokazuje się powiadomienie „You don't have access to view this file" (lub „[...] folder", w przypadku próby dostępu do katalogu). Jeżeli w analogiczny sposób użytkownik spróbuje dostać się do nieistniejącego elementu, powiadomienie będzie miało treść „This file could not be found" (lub „This folder [...]).

Przycisk służący do przenoszenia elementu do innego katalogu jest ukryty dla katalogu bazowego. Przetestowano przenoszenie plików – na liście katalogów do wyboru są jedynie katalogi należące do użytkownika. W przypadku katalogu „Child Directory" niemożliwe jest umieszczenie go wewnątrz katalogu „Grandchild Directory" – przejście wgłąb przenoszonego katalogu jest nieaktywne, jak pokazano na rys. 6.4.

Usunięto pojedynczy plik, a następnie katalog zawierający plik. W obydwu przypadkach operacja przebiegła pomyślnie.

W panelu powiadomień opcja dodania czasu wysłania powiadomienia zależnego od terminu wydarzenia lub zadania pojawia się tylko, jeśli termin ten został zdefiniowany. W każdym przypadku dostępna jest opcja zdefiniowania bezwzględnego czasu wysłania.



Rysunek 6.4: Zrzut ekranu z panelu przenoszenia katalogu „Child Directory”. Opcja przeniesienia katalogu do środka niego samego jest zablokowana.

Dodano powiadomienie, którego czas wysłania powinien być o 5 minut wcześniejszy, od terminu początkowego wydarzenia (w tym przypadku – godzina 18:55). Pojawiło się ono na liście nadchodzących powiadomień z czasem wysłania przeliczonym na bezwzględny – 18:50. O godzinie 18:50 przy nazwie użytkownika w pasku nawigacji pojawiła się cyfra „1” informująca o nowym nieodczytanym powiadomieniu. Powtórzono operację z takimi samymi ustawieniami. Liczba przy nazwie użytkownika natychmiast się zwiększyła, natomiast powiadomienie nie pojawiło się na liście powiadomień nadchodzących – nowo utworzone powiadomienie od razu jest wysłane.

Przetestowano również wysyłanie powiadomienia dla bezwzględnie określonego czasu. Sprawdzono zarówno dodawanie własnej wiadomości do powiadomienia, jak i pozostawienie treści domyślnej.

Na stronie profilu użytkownika wyświetlają się wszystkie wysłane powiadomienia. Oznaczyć je można jako odczytane i nieodczytane – sprawdzono, że odpowiednio zmieniana jest liczba przy nazwie użytkownika. Przetestowano również filtrowanie powiadomień według tego parametru.

Do pliku dodano powiadomienie, a następnie usunięto plik. Powiadomienie zostało kaskadowo usunięte.

W panelu głosowania na termin wydarzenia dodano kilka propozycji – formularz tworzenia obiektu `EventDate` jest widoczny tylko dla właściciela. Pole daty początkowej jest wymagane, obiekt nie zostanie też dodany, jeżeli data końcowa jest wcześniejsza od początkowej. Na propozycje zagłosowano, zmieniano wartości głosów, usuwano głosy: wynik całkowity zmienia się z każdą modyfikacją. Wydarzenie udostępniono dwóm innym użytkownikom – jednemu do edycji, drugiemu do wyświetlania. Obydwaj użytkownicy mogli zagłosować i dodać powiadomienia.

Jako właściciel wydarzenia usunięto jedną z propozycji, a inną zatwierdzono – termin wydarzenia odpowiednio się zmienił.

Modyfikacja danych użytkownika w profilu ma podobne działanie i wymagania, co podczas rejestracji. Aby zmienić hasło, należy podać aktualne oraz nowe hasło. Jeżeli po-

dane hasło jest nieprawidłowe, wyświetla się odpowiednie powiadomienie, pola formularza zawierają też weryfikację długości hasła. Po zmianie hasła wylogowano się i zweryfikowano, że zalogować można się tylko nowo ustalonym hasłem.

Przetestowano usuwanie konta użytkownika przez niego samego – operacja przebiegła pomyślnie.

Zalogowano się jako użytkownik posiadający prawa administratorskie. Na pasku nawigacyjnym wyświetlił się dodatkowy przycisk „Admin Panel". W panelu administratora sprawdzono działanie filtrowania, sortowania oraz stronicowania tabeli użytkowników. Kliknięcie na własnego użytkownika wyświetla jego dane. Kliknięcie na dowolnego innego użytkownika pokazuje ponadto przycisk nadawania lub odbierania praw administratora, w zależności od aktualnej roli danego użytkownika, oraz przycisk pozwalający na usunięcie użytkownika. Przetestowano operację zmiany uprawnień, każdorazowo logując się jako zmodyfikowany użytkownik i weryfikując, że stał się lub przestał być administratorem. Następnie usunięto jednego z użytkowników i sprawdzono, że został on usunięty z bazy danych.

Aplikację uruchomiono również w przeglądarkach Microsoft Edge wersji w 131.0 oraz Google Chrome w wersji 131.0. Sprawdzono działanie operacji takich jak logowanie i rejestracja, dodawanie i udostępnianie plików, dodawanie powiadomień, głosowanie na terminy. Nie zauważono różnic w działaniu aplikacji.

6.3 Wykryte i usunięte błędy

Autentykacja: podczas logowania generowane są dwa ciasteczka: `organizerJwt` oraz `organizerRefreshJwt`. Drugie z ciasteczek służy do odświeżania pierwszego, powinno więc mieć dłuższy czas ważności. Przez przypadek do obydwu ciasteczek przypisywano tę samą wartość `maxAge`.

Tworzenie katalogu bazowego przy rejestracji: podczas tworzenia katalogu sprawdzane jest, czy jego katalog nadrzędny istnieje. Nie uwzględniono sytuacji wyjątkowej, w której tworzony jest katalog bazowy nieposiadający katalogu nadrzędnego.

Odświeżanie ciasteczek: odświeżane było jedynie ciasteczko `organizerJwt` zamiast obydwu.

Tworzenie elementu: dodano obsługę sytuacji, gdzie katalog nadrzędny o podanym ID nie istnieje lub należy do innego użytkownika.

Pola w tabeli typu `TIMESTAMP`: typ danych `TIMESTAMP` w MySQL obsługuje jedynie przedział 1970-01-01 00:00:01 UTC do 2038-01-19 03:14:07 UTC. Należało dodać zabezpieczenia, przed podaniem daty spoza tego przedziału.

Usuwanie elementu: element powinien móc usunąć jedynie jego właściciel. Przypadkowo zrealizowano sytuację odwrotną: wszyscy użytkownicy niebędący właścicielem mogli usuwać elementy.

Panel dostępu do elementów: obiekty `AccessFile` lub `AccessDirectory` oraz obiekty `User` były przechowywane w osobnych tablicach. Obiekty użytkowników były w innej kolejności w tablicy, niż obiekty dostępów – błędnie założono, że dodawanie użytkowników iterując po obiektach dostępów spowoduje, że kolejność zostanie zachowana. W wyniku tego błędu, zmiany w przydzielonych dostępach następowały nie dla tych użytkowników, których nazwy były wyświetlone w liście w interfejsie graficznym panelu. Błąd naprawiono przechowując pary obiektów we wspólnej tablicy.

Rozdział 7

Podsumowanie i wnioski

7.1 Podsumowanie wykonanej aplikacji

W ramach projektu powstała aplikacja webowa wykorzystująca bazę danych, która pomaga użytkownikom w planowaniu zadań i wydarzeń. W tym celu wykorzystano technologie MySQL, Spring i Angular do stworzenia odpowiednio bazy danych, back-endu i front-endu. Aplikacja pozwala na tworzenie i udostępnianie notatek, wydarzeń i zadań, dodawanie do nich powiadomień, grupowanie ich w katalogach. Wyróżnia się funkcjonalnością głosowania na terminy wydarzeń. Jej interfejs nastawiony jest na zrozumiałość i łatwość użycia, a hasła użytkowników przechowywane są w bezpieczny sposób.

Na początku zaprojektowano relacyjną bazę danych. Niektóre tabele odpowiadają występującym w temacie pojęciom – użytkownik, plik, katalog. Inne, takie jak tabele przypisujące dostępy do elementów użytkownikom, bądź tabela zawierająca dane niezbędne do odświeżenia ciasteczek, są niezbędne do prawidłowego działania aplikacji, lecz niewidoczne dla jej użytkownika.

Implementację back-endu aplikacji utrzymano w konwencji frameworku Spring. Korzystając z mapowania obiektowo-relacyjnego tabele bazy danych zostały odwzorowane jako klasy języka Java. Dla każdej z nich zaimplementowano również klasy `Repository`, `Service` oraz `Controller`, a dla większości także klasy `DTO` oraz `Mapper`. Utworzono również klasy kluczy złożonych, konieczne do implementacji klas w relacji $N : M$, oraz pomocnicze klasy wyjątków, zwiększające czytelność kodu odpowiedzialnego za sprawdzanie dostępu do plików i katalogów.

Do implementacji dziedziczenia klas `Event`, `Note` oraz `Task` po klasie `File` w bazie danych wykorzystano metodę *Single Table Inheritance*, przez co w bazie odpowiada im wszystkim jedna tabela. Dzięki zastosowaniu dziedziczenia, operacje odczytu oraz usunięcia wierszy wystarczyło zaimplementować raz, dla klasy bazowej. Operacje tworzenia oraz aktualizacji wierszy musiały jednak być napisane osobno dla każdej klasy.

Konieczne było również zaimplemenowanie autentykacji i autoryzacji użytkowników.

Wykorzystano do tego framework Spring Security, który pozwolił na logowanie i wylogowywanie, oraz autoryzację ról użytkowników na poziomie poszczególnych metod.

We front-endzie odwzorowano klasy back-endu jako interfejsy języka TypeScript. Serwisy frameworku Angular zostały wykorzystane do komunikacji z back-endem. Konieczne było zadbanie, aby wraz z każdym zapytaniem przekazywały one ciasteczka autentykacyjne.

Zgodnie z konwencją frameworku, funkcjonalności aplikacji rozdzielono pomiędzy komponenty. Angular pozwala na tworzenie aplikacji typu Single-Page, wykorzystano jednak różne adresy, aby pozwolić użytkownikowi na zapisywanie poszczególnych adresów w zakładkach przeglądarki. Wiązało się to z dodatkowym problemem zapewnienia ciągłości stanu aplikacji, który rozwiązano używając pamięci sesji oraz pamięci lokalnej przeglądarki.

Większość komponentów w trakcie inicjalizacji pobiera dane z back-endu i pozwala użytkownikowi na edytowanie ich. Back-end wysyła zaktualizowane obiekty w odpowiedzi na zapytania, jednak konieczna była równoległa edycja danych już pobranych we front-endzie, aby zapewnić płynne działanie. Weryfikacja poprawności danych musiała więc zajść w obydwu częściach aplikacji.

Zapewniono bezpieczeństwo danych użytkownika: hasło przesyłane jest w zapytaniu z front-endu tylko podczas rejestracji i logowania, nigdy nie jest przesyłane w odpowiedzi. W bazie danych jest przechowywane w postaci zakodowanej, wykorzystując enkoder BCrypt.

W zapewnieniu intuicyjności aplikacji pomogło skorzystanie z biblioteki komponentów Angular Material. Dzięki temu wygląd interfejsu i poszczególnych kontrolek jest podobny do innych, znanych aplikacji. Oprócz tekstu wykorzystano wiele ikon, dodatkowo wskazujących na działanie każdego przycisku. Na przyciski, które, ze względów estetycznych, mają postać ikony bez tekstu, można najechać myszą, aby wyświetlić wskazówkę dotyczącą ich działania.

Przetestowano endpointy back-endu przy pomocy programu Postman. Sprawdzono zarówno przypadki poprawnych danych wejściowych, jak i dane, dla których zwracany jest kod błędu. Podczas testowania front-endu sprawdzono, czy dane są poprawnie przekazywane między front-endem a back-endem i poprawnie reprezentowane w interfejsie. Przetestowano sposoby reakcji interfejsu na błędy i niedozwolone operacje.

Projekt jest gotowy na dalsze rozszerzanie – do bazy danych można dodać tabele bez znacznego wpływu na napisany już kod back-endu, front-end podzielony jest na komponenty o dużym stopniu niezależności. Potencjalnym kierunkiem rozwoju może być dodanie funkcjonalności związanych z e-mailem. W obecnej aplikacji jest to opcjonalne pole, które nie jest wykorzystywane. Na adres e-mail mogłyby być wysyłane powiadomienia, przypominające użytkownikowi o ważnych terminach nawet wtedy, gdy sama aplikacja jest zamknięta. Można by było również dodać opcję odzyskiwania zapomnianego hasła poprzez wysłanie e-maila z jednorazowym kodem weryfikacyjnym.

Bibliografia

[1] *Angular*. 2024. URL: https://angular.dev/ (term. wiz. 20. 11. 2024).

[2] *Angular 17 JWT Authentication & Authorization example*. 2024. URL: https://www.bezkoder.com/angular-17-jwt-auth/ (term. wiz. 14. 12. 2024).

[3] *Angular 17 Refresh Token with JWT & Interceptor example*. 2024. URL: https://www.bezkoder.com/angular-17-refresh-token/ (term. wiz. 14. 12. 2024).

[4] *Angular Material UI component library*. 2024. URL: https://material.angular.io/ (term. wiz. 20. 11. 2024).

[5] *Angular v17 documentation*. 2024. URL: https://v17.angular.io/docs (term. wiz. 20. 11. 2024).

[6] *Docker Docs*. 2024. URL: https://docs.docker.com/ (term. wiz. 27. 11. 2024).

[7] *Entity Framework documentation hub*. 2024. URL: https://learn.microsoft.com/en-us/ef/ (term. wiz. 20. 11. 2024).

[8] *Luxon*. 2024. URL: https://moment.github.io/luxon/#/ (term. wiz. 21. 11. 2024).

[9] *Material Design*. 2024. URL: https://m2.material.io/ (term. wiz. 20. 11. 2024).

[10] *MySQL documentation*. 2024. URL: https://dev.mysql.com/doc/ (term. wiz. 20. 11. 2024).

[11] *Postman*. 2024. URL: https://www.postman.com/ (term. wiz. 01. 12. 2024).

[12] *React*. 2024. URL: https://react.dev/ (term. wiz. 20. 11. 2024).

[13] *Spring Boot Login example: Rest API with MySQL and JWT*. 2024. URL: https://www.bezkoder.com/spring-boot-login-example-mysql/ (term. wiz. 14. 12. 2024).

[14] *Spring Framework*. 2024. URL: https://spring.io/ (term. wiz. 20. 11. 2024).

[15] *Spring Security Refresh Token with JWT in Spring Boot*. 2024. URL: https://www.bezkoder.com/spring-security-refresh-token/ (term. wiz. 14. 12. 2024).

Dodatki

Spis rysunków