



Uniwersytet Rzeszowski
Kolegium Nauk Przyrodniczych
Instytut Informatyki

Praca projektowa programowanie obiektowe

Elektroniczny dziennik¹

Prowadzący:

mgr inż. Ewa Żesławska

Autor:

Tomasz Nowak

nr albumu: 131478

Kierunek: Informatyka, Grupa 1 Laboratorium 1

Rzeszów 2024

¹ Przykład poprawnie zdefiniowanego tytułu: System raportowania przewinień w biurze, Elektroniczny system oceniania, itp. Przykład niepoprawnie zdefiniowanego tytułu: Kuchenka mikrofalowa, Samochód, itp.

Spis treści

1.	Opis założeń projektu	3
2.	Opis struktury projektu	5
3.	Harmonogram realizacji projektu	12
4.	Prezentacja warstwy użytkowej projektu	14
5.	Podsumowanie	19
6.	Literatura	21

1. Opis założeń projektu

W tym rozdziale opisane są założenia projektu 'Elektroniczny Dziennik'

- Celem projektu jest stworzenie aplikacji 'Elektroniczny Dziennik', która będzie dostępna do użycia dla, każdego użytkownika, który posiada łącze internetowe oraz podstawowy komputer z systemem operacyjnym Linux/MacOS/Windows.
- Problemem, który będzie rozwiązywany przez aplikację "Elektroniczny Dziennik", jest skuteczne zarządzanie ocenami studentów. Podstawowym źródłem tego problemu jest tradycyjny, papierowy system dzienników, który jest nieefektywny oraz trudny w zarządzaniu dla oceniającego jak i ocenianego. Największym problemem jest przede wszystkim brak dostępności zdalnej do wpisów w dzienniku.
- Niezbędnym do rozwiązania problemu jest stworzenie aplikacji, która posiada system uwierzytelniania dla oceniającego jak i ocenianego, również ważnym jest, aby oceny były dostępne tylko dla użytkowników upoważnionych do ich wyświetlania oraz edytowania bądź usuwania.
- Do rozwiązania problemu zostanie stworzony interfejs użytkownika, który będzie się komunikował z serwerem e-dziennika za pomocą plików JSON.

Wymagania funkcjonalne

- System logowania do systemu.
- Autoryzacja użytkownika przy każdym wysyłanym żądaniu do serwera za pomocą Tokenu nadawanemu przy autentykacji użytkownika.
- Panele do autentykacji poszczególnych użytkowników, aby następnie przekierować uwierzytelnionego użytkownika do odpowiedniego panelu użytkownika.
- Prosty widok wszystkich ocen użytkownika ocenionego (w domyśle studenta)
- Klarowny i prosty interfejs użytkownika zezwalający na operacje tworzenia, edytowania, usuwania ocen przez oceniającego (w domyśle prowadzący lub nauczyciel).
- Bezpieczeństwo danych użytkowników - odpowiednia autoryzacja dostępu poszczególnych użytkowników. Oceniający mają widzieć tylko wystawione przez siebie oceny, natomiast student ma widzieć tylko swoje oceny, które są wystawione na jego numer indeksu.
- Baza danych stworzona w systemie zarządzania MySQL na naszym serwerze.

- Poprawnie działający Connector z naszą bazą danych.
- Połączenie serwera z bazą danych na tym samym serwerze lub innym (przykładowo RDS) do przetwarzania danych.
- Serwer obsługujący zapytania użytkownika w formie HTTP Requests (Get, Post, Delete, Update) wysyłane bezpośrednio z interfejsu użytkownika. Serwer powinien zwracać odpowiednio przygotowane dane z utworzonej bazy danych w formie JSON.
- Użytkownik, który stworzył konto studenckie nie może zalogować się nim w panelu nauczyciela. Obsługa tego wyjątku.

Wymagania niefunkcjonalne

- Wydajność - Aplikacja IT powinna działać na jak najprostszych urządzeniach obsługujących Javę.
- Środowisko aplikacji powinno działać na każdym popularnym systemie operacyjnym.
- Niezawodność aplikacji, odporność na błędy użytkowników.
- Aplikacja IT powinna mieć klarowny design dla osób daltonistycznych.
- Utrzymywalność - dokumentacja projektu, rozdzielenie Back-endu i panelu klienta na rzecz łatwiejszej rozbudowy aplikacji.
- Jedno poziomowe uwierzytelnianie.
- Aplikacja powinna mieć ustawiony Font przycisków na Courier New.
- Aplikacja IT powinna płynnie obsługiwać ilość użytkowników na poziomie około 10 tysięcy w skali dnia, przy założeniu, że jeden serwer z aplikacją back-endową odpowiada jednej uczelni.
- Aplikacja IT powinna powiadamiać użytkownika o nieudanym logowaniu
- Aplikacja IT powinna powiadamiać użytkownika o poprawnym logowaniu.
- Dashboard użytkownika powinien się pojawić po udanej autoryzacji użytkownika (przejsie do Dashboard po kliknięciu w oknie dialogowym przycisku OK)
- Użytkownicy powinni ustawiać swoje indeksy przy pierwszym logowaniu na danego E-maila
- Efektywność kosztowa aplikacji

2. Opis struktury projektu

W tym rozdziale zostanie opisana struktura całego projektu. Stworzenie Elektronicznego dziennika rozdzielono na dwie niezależne od siebie aplikacje, jest to interfejs użytkownika oraz serwer z aplikacją w strukturze REST API. Cały system został rozdzielony na potrzeby utrzymania stabilności całego projektu oraz jego stopniowego bezproblemowego rozwijania.

Opis aplikacji działającej na serwerze

Do stworzenia aplikacji back-endowej użyto Javy z frameworkiem Spring, gdyż bardzo dobrze on współgra z Hibernate czyli frameworkiem do obsługi naszej bazy danych. Hibernate zapewnia nam mapowanie obiektowo-relacyjne (ORM), co bardzo przyspiesza tworzenie oraz rozwijanie naszej aplikacji back-endowej. Po odpowiednim skonfigurowaniu naszej aplikacji Spring mamy pełną możliwość korzystania z tego narzędzia.



Rysunek 1. Diagram klas zaprojektowanej aplikacji działającej na serwerze (Obrazek dostępny również w repozytorium)

Przyjrzymy się trzeciej kolumnie na Rysunku 1. Wyróżnimy w tej kolumnie 3 klasy i jeden interfejs. Cała ta kolumna odpowiada za działanie całego systemu oceniania.

1. GradeRepository - ten interfejs rozszerza interfejs JpaRepository który jest odpowiedzialny za operacje CRUD (Create, Read, Update, Delete), jak i również paginację, sortowanie, tworzenie kwerend w naszej bazie danych. 'GradeRepository' rozszerza wyżej opisany interfejs, gdzie w ostrych nawiasach podajemy mu naszą klasę Modelu (Entity) którą ma zarządzać oraz typ identyfikatora tej klasy.
2. Grade – jest to klasa naszego modelu oceny reprezentowanej w bazie danych. Dzięki modelowi nasze Repository może bezproblemowo stworzyć daną tabelę w bazie danych. Nasz model posiada takie dane jak: ocena, indeks ucznia, przedmiot, opis, nauczyciel wystawiający ocenę, imię i nazwisko studenta.
1. GradeService – klasa, którą już stricte działamy w kontrolerach naszej aplikacji, to do niej przesyłamy zmienne, które 'GradeRepository' ma przetwarzać. Jest to można powiedzieć taki 'wrapper' naszego repozytorium, w którym możemy implementować logikę biznesową.
2. GradeController – klasa odpowiedzialna za warstwę webową naszej aplikacji, to tutaj wpada nasz cały ruch sieciowy który wysyła żądanie do naszego serwera na ścieżkę "/api/grades/". Ta klasa jest traktowana jako kontroler REST przez Spring, co oznacza, że metody w tej klasie będą obsługiwać żądania HTTP i zwracać dane za pomocą plików JSON. W tym kontrolerze zajmujemy się filtracją żądań między innymi GET/POST/PUT/DELETE. Ta klasa zwraca klientowi dane z np. 'GradeService', z konkretnym statusem odpowiedzi np. 200, 404, 203.
 - a. W tym kontrolerze zastosowano autoryzację odnośnie do wszelakich działań z ocenami (Tworzenie/edytowanie/usuwanie/czytanie) za pomocą konta użytkownika, po rozstrzygnięciu do jakiego użytkownika należy żądanie (za sprawą autoryzacyjnego nagłówka z nadanym wcześniej tokenem w żądaniu), sprawdzamy czy dany użytkownik jest nauczycielem w przypadku tworzenia/edytowania/usuwania/zobaczenia ocen. W przeciwnym przypadku, gdy użytkownik jest studentem, ma on możliwość tylko zobaczenia swoich ocen, wszelkie próby zmiany ocen przez studenta (Wysyłane bezpośrednio do serwera, gdyż interfejs użytkownika nie zezwala na działania dodawania/edytowania/usuwania oceny w panelu studenta) zostają zignorowane przez serwer statusem "unauthorized".

Pierwsza kolumna oraz kolumna środkowa odpowiadają za autoryzację użytkownika

3. `JwtAuthorizationFilter` – ta klasa służy nam za wstępny filtr, tutaj wpadają nasze żądania sieciowe które następnie trafiają do właściwego filtra `'JwtUtil'` który sprawdza poprawność żądania.
4. `JwtUtil` – ta klasa sprawdza nam zawarty w żądaniu nagłówek autoryzacyjny w którym jest nadany token przy logowaniu (zakładając, że użytkownik zalogował się pomyślnie) - Mała uwaga dla pisania chociażby serwisu WWW czy innego interfejsu klienta - interfejs użytkownika powinien ustawiać nagłówek autoryzacyjny z tokenem o przedrostku `'Bearer xxx'` gdzie `'xxx'` to token zwracany przez backend przy pomyślnym logowaniu.
5. `SecurityConfig` – w tej klasie konfiguracyjnej dla Springa, włączamy nasz `JwtAuthorizationFilter`, aby poprawnie odsiewać niechciane dla nas wszelkie żądania HTTP.
6. `AuthController` – w tej klasie tworzymy logikę webową naszego całego przebiegu autentykacji, `AuthController` jest zmapowany jako REST controller do ścieżki `"rest/auth/login"`, w tym kontrolerze tworzymy konto nauczyciela bądź ucznia w zależności od subdomeny w podanym e-mailu przez użytkownika. Konto zostanie stworzone, jeśli wcześniej nikt nie zajął tego e-maila. Uwaga! Jako iż jest to aplikacja na potrzeby nauki, to hasła nie są szyfrowane w żaden sposób więc domyślne hasło dla każdego użytkownika to `'123'` (nawet gdy użytkownik przy pierwszym logowaniu poda swoje hasło). W przyszłości planowane jest rozwinięcie tego kontrolera o nową ścieżkę `"rest/auth/register"`, gdzie będzie się odbywać poprawna rejestracja konta z podanym hasłem użytkownika.
7. `User` (entity), `CustomUserDetailsService`, `UserRepository` – Tworzenie systemu kont użytkowników, powyżej opisano dokładnie działanie `"service"`, `"repository"`, `"model"`. Różnicą jest zmiana przetrzymywanych danych oraz ich obsługa.
8. `LoginReq`, `ErrorRes`, `LoginRes` - są to modele, które przyjmują i zwraca nasz `AuthController`. JSON wysyłany przez interfejs użytkownika jest rzutowany na obiekt który zdefiniowaliśmy jako `LoginReq` itp.

Opis aplikacji działającej na komputerze użytkownika

Do stworzenia interfejsu użytkownika użyto Javy z frameworkiem JavaFX, jest to dużo świeższe narzędzie do tworzenia interfejsów niż Swing, stąd padł wybór na JavaFX. Tworzenie interfejsu w tym frameworku jest w miarę intuicyjne oraz proste. Zaleca się do tworzenia nowych widoków bądź edycji już istniejących program Scene Builder. Do stworzenia aplikacji webowej zalecane jest użycie narzędzi stricte webowych tj. HTML, CSS, JavaScript w różnych ‘opakowaniach’ - między innymi React, NextJS + React itp. Dzięki serwerowi z aplikacją, która obsługuje REST API, można śmiało rzec “Sky is the limit”.



Rysunek 2. Diagram klas zaprojektowanej aplikacji desktopowej (Obrazek dostępny również w repozytorium)

Możemy śmiało zauważyć na Rysunku 2, że występują tutaj dwa “typy” klas, jedno z nich zawierają w nazwie “controller” a drugie są komponentami, gdzie każdy z tych komponentów za coś konkretnie odpowiada. Podążając za zasadami SOLID, każdy komponent musi odpowiadać za stricte jedną rzecz, jest to zasada pojedynczej odpowiedzialności. Zaczniemy od omawiania klas kontrolerów

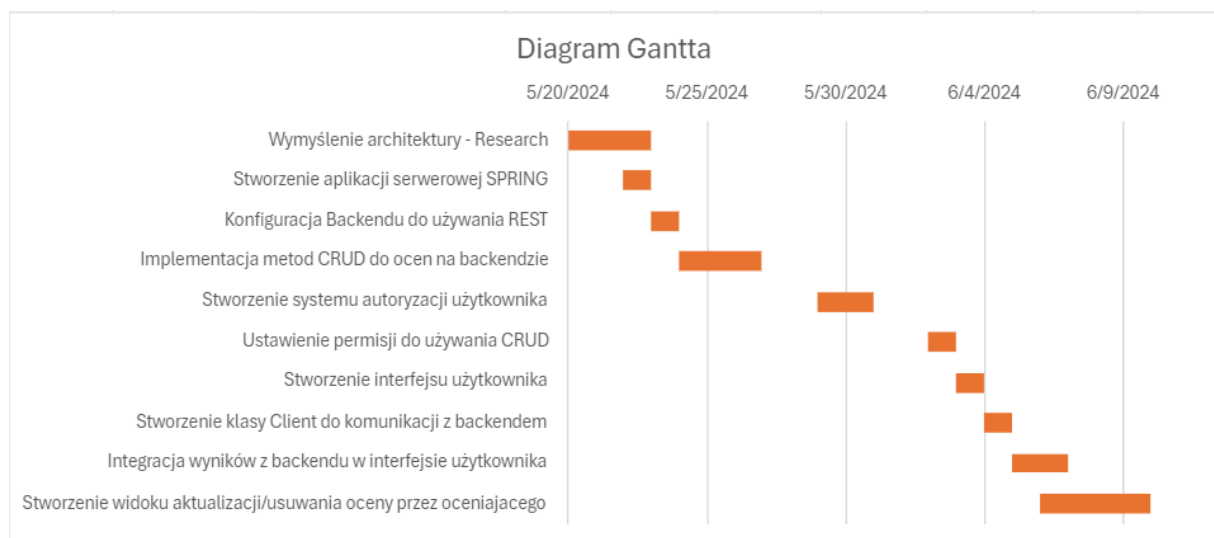
1. PrimaryController – jest to kontroler, do którego trafia użytkownik zaraz po włączeniu aplikacji, z jego widoku użytkownik może przenieść się do panelu logowania studenta/ucznia bądź prowadzącego/nauczyciela.
2. StudentController – jest to kontroler, do którego trafia student/uczeń z widoku ‘Primary’, dzięki niemu użytkownik jest w stanie zalogować się wyłącznie jako student. W tym kontrolerze niemożliwym jest zalogowanie się poprzez inny email niż ten, który jest subdomeną stud.urz.pl. Po pomyślnym zalogowaniu bądź nie, wyświetli się stosowny komunikat.
3. TeacherController – kontroler, do którego trafia prowadzący/nauczyciel z widoku ‘Primary’, dzięki niemu użytkownik jest w stanie zalogować się tylko jako oceniający. Niemożliwym jest zalogowanie się za pomocą innego emaila niż ten zawierający domenę urz.pl. Po pomyślnym zalogowaniu bądź nie, wyświetli się stosowny komunikat.
4. StudentPanelController – kontroler z widokiem na wszystkie oceny ucznia. Oceny są filtrowane po stronie serwera za pomocą tokenu. W tym widoku jedyne co uczeń może to zobaczyć swoje oceny. Nie da się w żaden sposób dodawać/edytować/usuwać oceny, ponieważ autoryzacja do tych czynności jest sprawdzana po stronie serwera. Użytkownik może na bieżąco ściągać oceny z serwera za pomocą dedykowanego przycisku.
5. TeacherPanelController – kontroler z widokiem na wszystkie wystawione oceny przez prowadzącego. Niemożliwym jest, aby nauczyciele widzieli oceny wystawione przez innych nauczycieli. Użytkownik nauczyciela może w prosty sposób dodawać/edytować/usunąć ocenę (po kliknięciu na daną ocenę otworzy się panel edycji), Użytkownik może na bieżąco ściągać oceny z serwera za pomocą dedykowanego przycisku. Możliwe jest sortowanie ocen (kliknąć nazwę kolumny dla innego sortowania), gdzie sortowanie odbywa się po stronie komputera użytkownika, optymalizując działanie serwera poprzez mniejszą ilość zapytań.

6. GradeDetailController - kontroler, który wyświetla szczegóły naszej oceny, w tym kontrolerze możemy aktualizować lub usuwać naszą ocenę.
7. AlertPopUp – klasa do pokazywania wszelakich alertów użytkownikowi.
8. EmailValidator – klasa zawierająca metody do sprawdzania domen w podawanych przez użytkowników emailach (domeny są potrzebne, aby zalogować użytkownika do serwera, jak i również wyświetlenia stosownego komunikatu, gdy domena nie zgadza się z panelem logowania tzn. Gdy użytkownik loguje się w panelu studenta za pomocą subdomeny stud.urz.pl i na odwrót). Wyniki tych funkcji tej klasy są później obsługiwane przez kontrolery z logowaniem.
9. Client – klasa umożliwiająca nawiązanie połączenia z naszym serwerem. Założeniem tej klasy jest to, aby każda klasa typu kontroler i nie tylko, korzystała z tego samego “Client’a”, gdyż w nim po udanym logowaniu zapisujemy Bearer Token w nagłówku autoryzacji, gdzie później służy nam on do wysyłania autoryzowanych żądań do serwera. Bez tokenu każde żądanie wysłane do serwera nie powiedzie się (każde zapytanie do ścieżki innej niż “rest/auth/login”).
 - a. Metoda authorizeClient – to w tej metodzie autoryzujemy i zarazem autentykujemy nasz komponent klienta. Dostępne są dwie metody o tej samej nazwie, jedna posiada parametry dla logowania studenckiego, druga dla logowania oceniającego. To w tej metodzie wysyłamy żądanie POST do backendu na ścieżkę “rest/auth/login”, jeśli odpowiedź od serwera zawiera token autoryzacyjny to zostaje wywołana funkcja z ustawieniem naszego Bearer Tokenu dla klienta.
 - b. Metoda getListRequest – metoda zwracająca ArrayListę z obiektami typu Object. Metoda przyjmuje wartość “path” czyli ścieżkę, na którą metoda ma wysłać zapytanie typu GET. Całą odpowiedź od serwera rzutujemy na typ ArrayList<Object>, która jest zwracana z tej metody. Używamy głównie tej metody do pobierania wszystkich przykładowo ocen, itp.
 - c. Metoda postRequest – metoda przyjmująca wartości “path” (ścieżka, na którą ma zostać wysłane żądanie typu POST), oraz HashMap<String, Object> - HashMapa zawierająca klucze typu String z wartościami typu Object. W tej hashmapie powinny być dane, które chcemy wysłać na serwer. Ta hashmapa zostanie zmieniona na obiekt typu JSON i zostanie wysłana na serwer z uwzględnieniem naszego autoryzacyjnego tokenu.

- d. Metoda `putRequest` – metoda przyjmująca te same wartości z tymi samymi własnościami co metoda `postRequest`, jedyną zmianą jest to, iż musimy uwzględnić w ścieżce ID oceny, którą chcemy zmienić. W przypadku aktualizacji oceny posiadającej ID=3, nasz parametr ścieżki powinien wyglądać w następujący sposób: `"/api/grades/3"`. Metoda wyśle żądanie do serwera typu PUT na powyżej podaną ścieżkę z wcześniejszym przygotowaniem parametru HashMapy na obiekt JSON, oraz uwzględnieniem naszego tokenu.
 - e. Metoda `deleteRequest` – metoda przyjmująca ścieżkę z parametrem ID oceny, która ma być usunięta. Np: Jeśli chcemy usunąć ocenę z ID=5 to parametr naszej ścieżki powinien wyglądać w następujący sposób: `"/api/grades/5"`. Po odpowiednim określeniu ścieżki zostanie wysłane żądanie do serwera typu DELETE z wcześniejszym ustawieniem tokenu autoryzacyjnego do naszego żądania.
10. `Grade` – klasa do reprezentacji oceny w JavaFX TableView. Ściągając dane z serwera rzutujemy je do tej klasy. Klasa ta posiada tylko zmienne do reprezentacji danych wraz z domyślnymi getterami/setterami.

3. Harmonogram realizacji projektu

Harmonogram realizacji projektu umieszczono poniżej. Najbardziej problematycznym było skonfigurowanie całego projektu w Spring w związku z brakiem doświadczenia w tworzeniu w tym konkretnym frameworku. Po odpowiedniej konfiguracji i odpowiednim researchu całość prac była już tylko przyjemną formalnością.

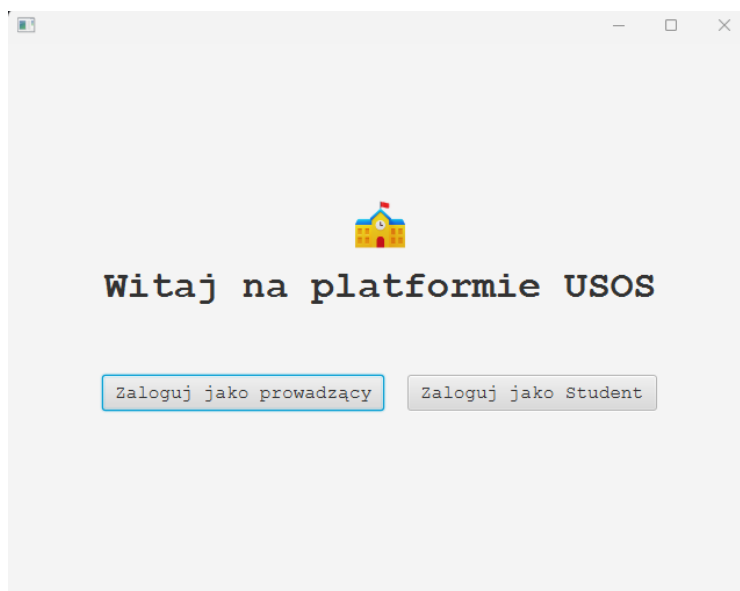


Rysunek 3. Diagram Gantta

Projekt realizowany był z wykorzystaniem systemu kontroli wersji Git, wszystkie pliki źródłowe projektu znajdują się pod adresem: <https://github.com/Tnovyloo/Java-class-register-app> i będą dostępne do 31.06.2025. Wszystkie commity i ich historia jest dostępna pod adresem: <https://github.com/Tnovyloo/Java-class-register-app/commits/main/> (w związku z dużą ilością commitów, nie zostanie umieszczone ich zdjęcie)

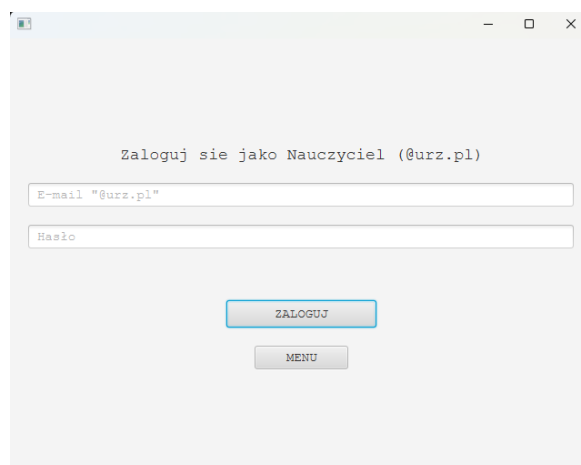
4. Prezentacja warstwy użytkowej projektu

Na Rysunku 1 przedstawiono główne okno aplikacji. W tym oknie użytkownik może wybrać na jakie konto chce się zalogować.



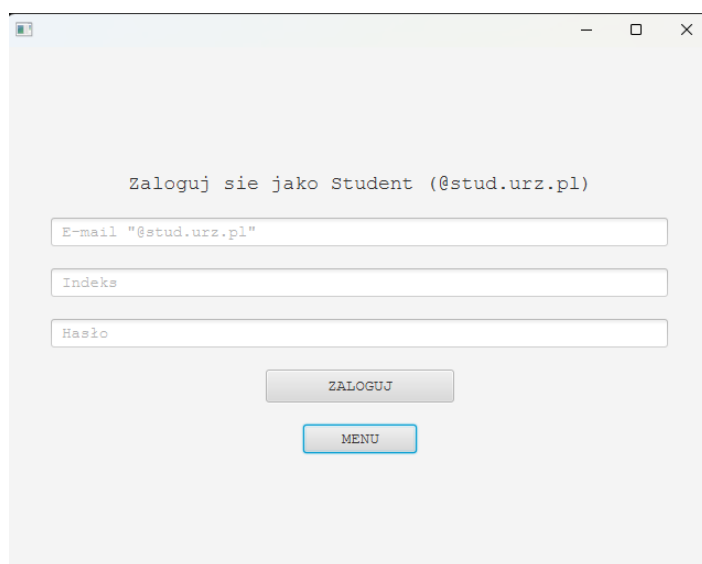
Rysunek 1 - Główne okno aplikacji

Na Rysunku 2 przedstawiono panel logowania na konto nauczyciela. Są tutaj tylko dwa TextInputy mianowicie: E-mail, oraz hasło. Gdy użytkownik loguje się pierwszy raz jako nauczyciel/prowadzący to tworzy nowe konto na adres e-mail gdy ten nie jest powiązany z żadnym innym kontem. Użytkownik może użyć wyłącznie e-maila który posiada w sobie domenę urz.pl. Przycisk menu przekierowuje nas do głównego panelu aplikacji.



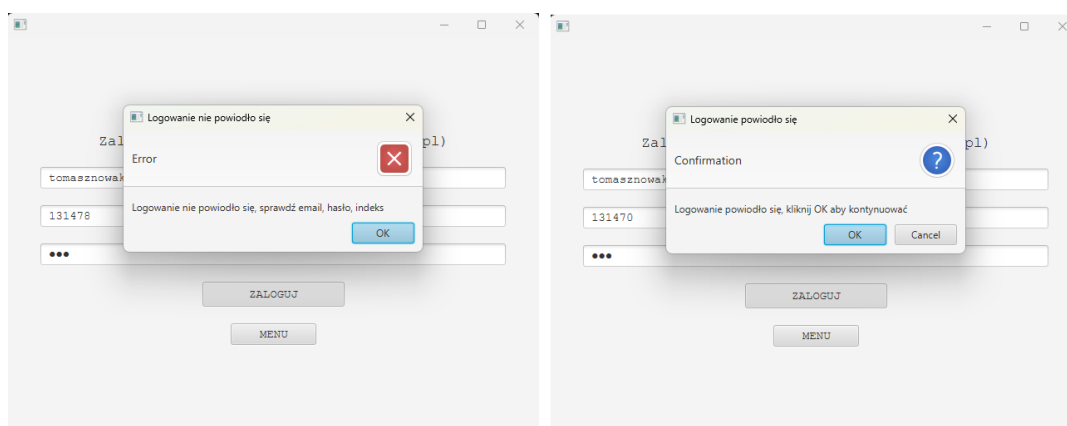
Rysunek 2 – Okno logowania dla nauczyciela/prowadzącego

Na Rysunku 3 przedstawiono panel logowania dla studenta bądź ucznia. Różnicą jest jeden TextInput więcej, zawiera on pole tekstowe z indeksem. Przy pierwszym logowaniu na konto studenta użytkownik przypisuje do podanego adresu e-mail swój indeks. Adres e-mail musi zawierać w sobie subdomenę stud.urz.pl. Uwaga! Niemożliwym jest zalogowanie na konto studenckie z podaniem nieprawidłowego indeksu, tzn. Użytkownik chcąc zalogować się na swoje konto musi podawać indeks który podał przy pierwszym logowaniu, inaczej back-end zignoruje żądanie logowania i zwróci odpowiedź 'Bad request'.



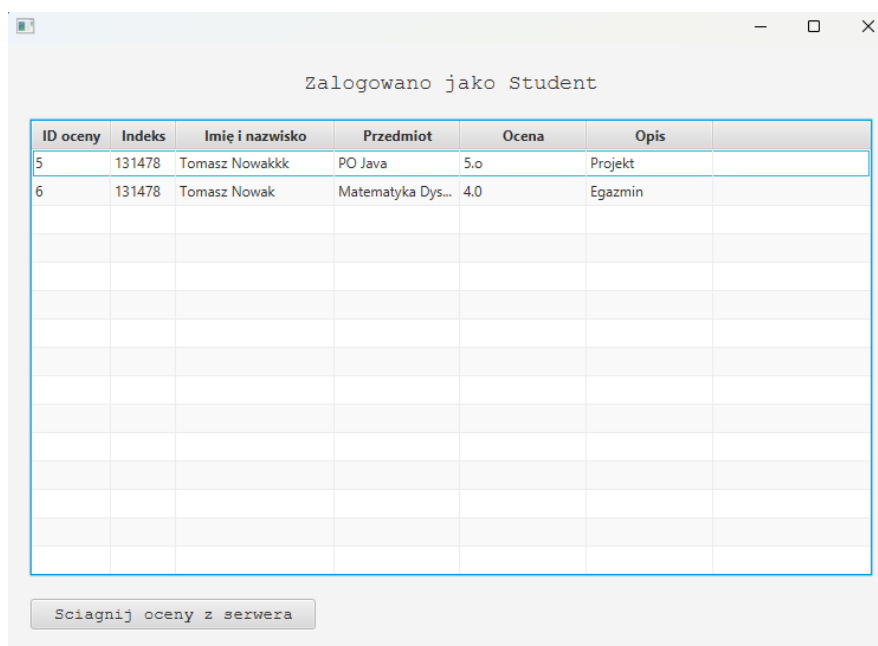
Rysunek 3 – Panel logowania studenta

Na Rysunku 4 przedstawiono sytuację poprawnego logowania jak i błąd w autentykacji. W panelu logowania nauczyciela również są uwzględnione alerty.



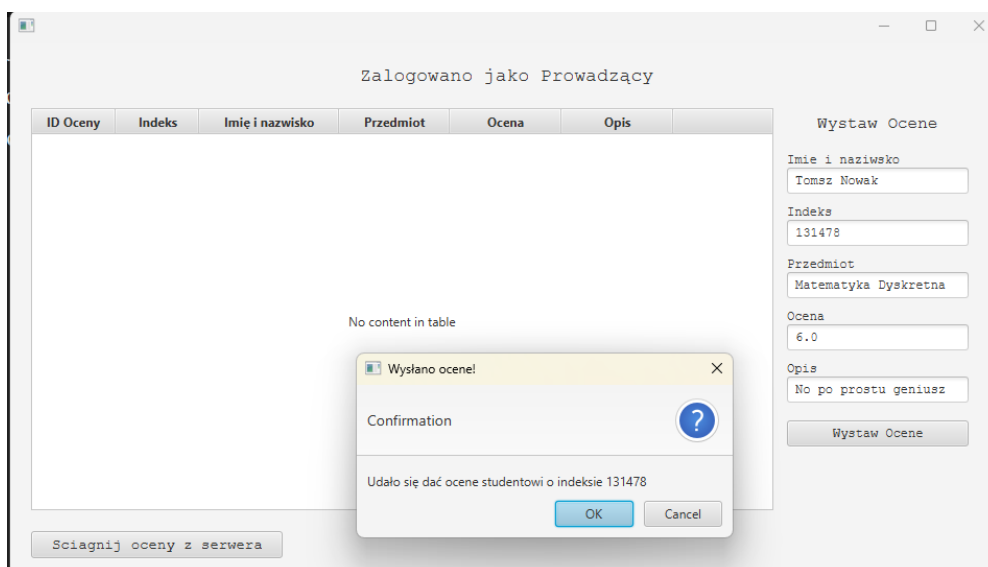
Rysunek 4 – Alerty przy logowaniu.

Na Rysunku 5 widoczny jest panel studenta, wyświetla się on tylko po poprawnym zalogowaniu. Klikając na kolumny w naszej tabeli możemy sortować nasze wyniki. Użytkownik widzi wyłącznie swoje oceny i nie ma żadnej możliwości zobaczenia ocen innej osoby gdyż są one odpowiednio filtrowane po stronie serwera. Panel studenta ma tylko jeden przycisk do odświeżenia ocen z serwera. Po jego kliknięciu klasa Client wyśle ponownie zapytanie o oceny użytkownika i wstawi je do tabeli użytkownika.



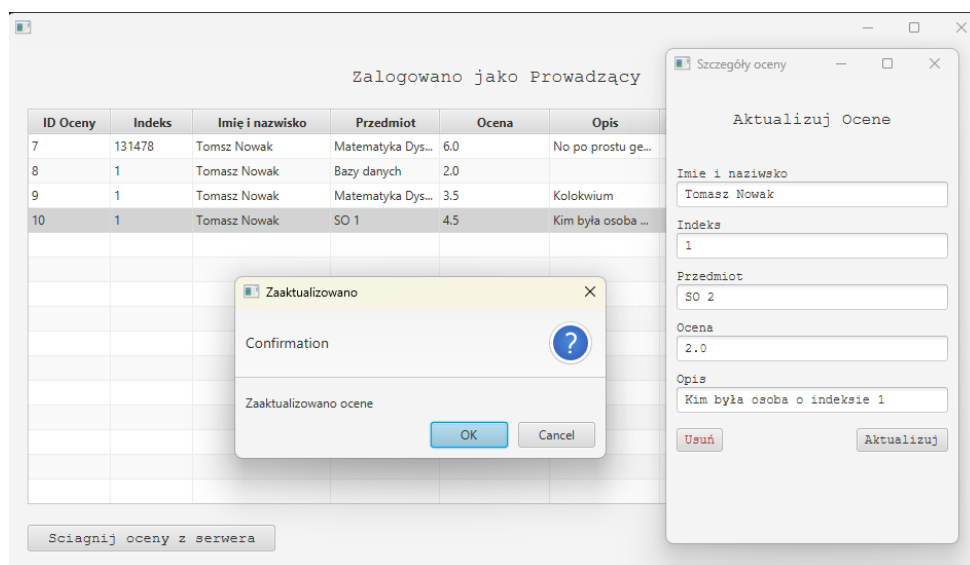
Rysunek 5 – Panel studenta

Na Rysunku 6 przedstawiono panel nauczyciela, w którym użytkownik wystawił ocenę. Jak widać w tym panelu użytkownik może wystawić ocenę, edytować bieżące oceny, które wcześniej wystawił. Co ważne, użytkownik oceniający widzi tylko oceny wystawione przez siebie. Odpowiednia filtracja dzieje się po stronie serwera stąd nie ma tutaj mowy o żadnym przecieku danych.



Rysunek 6 – Panel nauczyciela, w którym wystawiono ocenę.

Rysunek 7 przedstawia panel edycji oceny, który otwiera się w nowym oknie po jednokrotnym kliknięciu na dany wiersz z oceną w naszej tabeli. Mamy w nim dwa przyciski tj. 'Aktualizuj' oraz 'Usuń'. Jak jest widoczne na Rysunku 7 została edytowana ocena o ID równym 10. Zmieniono w niej ocenę oraz przedmiot. Dodatkowo poza kulisami Rysunku 7 zostanie usunięta ocena o ID równym 7. Wyniki działania aktualizacji oraz usuwania ocen po pobraniu ocen nauczyciela na nowo z back-endu mają następujący efekt na Rysunku 8.



Rysunek 7 – Panel nauczyciela z widocznym oknem edycji oceny

Zalogowano jako Prowadzący

ID Oceny	Indeks	Imię i nazwisko	Przedmiot	Ocena	Opis
8	1	Tomasz Nowak	Bazy danych	2.0	
9	1	Tomasz Nowak	Matematyka Dys...	3.5	Kolokwium
10	1	Tomasz Nowak	SO 2	2.0	Kim była osoba ...

Wystaw Ocene

Imię i nazwisko

Indeks

Przedmiot

Ocena

Opis

Rysunek 8 – Panel nauczyciela po edytowaniu oceny z ID równym 10 oraz usunięciu oceny z ID równym 7.

Teraz zostanie przedstawiona symulacja elektronicznego dziennika w warunkach “rzeczywistych”. Rysunek 9 oraz Rysunek 10 przedstawia dwóch różnych nauczycieli, którzy wystawili ocenę studentom o numerach indeksów 111 oraz 222. Na Rysunku 11 przedstawiono panel studenta o numerze indeksu 111. Jak widać filtrowanie ocen na back-endzie działa pomyślnie i nie wyświetlają się temu studentowi oceny innych studentów.

Zalogowano jako Prowadzący

ID Oceny	Indeks	Imię i nazwisko	Przedmiot	Ocena	Opis
8	1	Tomasz Nowak	Bazy danych	2.0	
9	1	Tomasz Nowak	Matematyka Dys...	3.5	Kolokwium
10	1	Tomasz Nowak	SO 2	2.0	Kim była osoba ...
11	111	Nowak 111	Haskell	6.0	
12	222	Nowak 222	Haskell	2.0	Poprawa 20.06

Wystaw Ocene

Imię i nazwisko

Indeks

Przedmiot

Ocena

Opis

Rysunek 9 – Panel elektronicznego dziennika nauczyciela ‘A’

Zalogowano jako Prowadzący

ID Oceny	Indeks	Imię i nazwisko	Przedmiot	Ocena	Opis
13	111	Nowak 111	OOP TOP	5.0	Jaki LLM?
14	111	Nowak 111	OOP TOP	3.0	Projekt
15	222	Nowak 222	OOP TOP	5.0	Egzamin
16	222	Nowak 222	OOP TOP	3.0	Ćwiczenia

Wystaw Ocene

Imię i nazwisko

Indeks

Przedmiot

Ocena

Opis

Wystaw Ocene

Ściągnij oceny z serwera

Rysunek 10 – Panel elektronicznego dziennika nauczyciela ‘B’

[illegible]

Rysunek 10 – Panel studenta o numerze indeksu 111

5. Podsumowanie

Podsumowując, wykonane zostały dwie niezależne od siebie aplikacje. Oczywiście, żeby w pełni używać interfejs użytkownika potrzebny jest działający back-end. Założenie projektu zostało spełnione, lecz poniżej znajduje się lista rzeczy, którą będzie można na spokojnie w przyszłości wdrożyć do naszego systemu elektronicznego dziennika.

1. Lepszy system autentykacji uwzględniający rejestrację użytkownika w osobnym panelu. Do tego potrzeba byłoby stworzenia funkcji rejestrujących w AuthController na back-endzie.
2. Połączenie dwóch paneli logowania w jeden.
3. Dużo lepszy UI/UX - chociaż to mogłoby być zastąpione nową aplikacją webową.
4. Rozwinięcie panelu użytkowników o nowe funkcjonalności np: rysowania diagramów itp.
5. Dockeryzacja projektu Spring – pomimo prób, niestety uruchomienie aplikacji Spring na docker-compose i jednoczesny rozwój projektu lokalnie, było zbyt kłopotliwym zadaniem dla laika w Springu.
6. Stworzenie testów i wprowadzenie Continuous Integration w repozytorium projektu – do weryfikowania poprawności wprowadzanych zmian w kodzie aplikacji.
7. Stworzenie Continuous Deployment – w przypadku używania własnego serwera trzeba zainteresować się webhookami. Przy używaniu szeroko dostępnych drogich chmur obliczeniowych temat jest dużo prostszy do zrealizowania.

6. Literatura

1. Dokumentacja [JavaFX](#)
2. Dokumentacja [Spring](#)
3. Aqeel Abbas - Creating a Rest API in Spring Boot - <https://medium.com/javajams/creating-a-rest-api-in-spring-boot>
4. Farhan Tanvir - Spring Security JWT Authentication & Authorization <https://medium.com/code-with-farhan/spring-security-jwt-authentication-authorization>
5. Yiğit Kemal Erinç - SOLID Principles (Artykuł teoretyczno-praktyczny) - <https://www.freecodecamp.org/news/solid-principles>