# Część praktyczna

# Projekt aplikacji wspomagającej aranżacje muzyczne

## Architektura aplikacji

Jako model architektury aplikacji przyjęto architekturę warstwową. Pozwala ona odseparować zadania wykonywane przez poszczególne komponenty aplikacji, ułatwiając utrzymanie kodu. Jest to osiągane poprzez rozgraniczenie zadań realizowanych przez poszczególne warstwy. Architektura warstwowa charakteryzuje się hierarchicznym układem komponentów, co przekłada się na zwiększenie przejrzystości kodu. W ramach projektu zastosowano trzy warstwy podziału:

* Warstwa dostępu do danych — odpowiadająca za komunikacje z bazą przy użyciu modelu encji.
* Warstwa logiki biznesowej — obejmująca szczególnie serwisy zarządzania stanem aplikacji.
* Warstwa prezentacji — realizowana przez komponenty Vue.

## Warstwa dostępu do danych

### Modele bazodanowe

Dane użytkownika, zapisywane w aplikacji można sklasyfikować jako forma “notatek”. Zdecydowano się nadać im modułowy charakter, pozwalając w ten sposób na wykorzystywanie różnych typów notacji muzycznej w ramach określonego zbioru, który sumarycznie może opisywać aranżowany utwór. Wykorzystano stąd dwupoziomowy podział obiektów. Jako pierwszy poziom przyjęto koncepcję “zeszytu do nut”, jako zbioru notatek, co przynosi stosowną nazwę wykorzystaną w kodzie źródłowym “book” (od ang. music book — *zeszyt do nut*).

<encja zeszytu i opis pól>

Idąc dalej tym tokiem, przyjęto określenie “page” (z ang. — textitstrona), jako odnoszące się do poziomu drugiego — pojedynczej notatki użytkownika.

<encja książki i opis pól>

Obie encje implementują prosty interfejs Entity, deklarujący pole id, ułatwiając działanie serwisów zarządzających persystencją.

<interfejs entity>

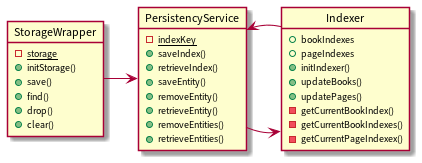
Jako wyznacznik powiązań między encjami Book i Page, w bazie przechowywana jest dotatkowo tablica obiektów Index. Z uwagi na jej nadrzędny charakter, nie implementuje ona od interfejsu Entity. Występuje w bazie pojedynczo, pod ustalonym kluczem “INDEX”.

<encja index>

### Serwisy zarządzające persystencją

W aplikacji zastosowano bazę danych Ionic Storage. Jest to baza typu klucz-wartość. Aby umożliwić zarządzanie obiektami zagnieżdżonymi — jako: *zeszyt* zawierać może wiele *stron* — przy zachowaniu niezależności operacji na obiektach, zastosowano trzy serwisy:

* StorageWrapper
* Z uwagi na charakter kluczy i wartości bazy — jako dwóch ciągów znaków — obiekty zapisywane w bazie są konwertowane do ciągów JSON. StorageWrapper *opakowuje* bibliotekę Ionic Storage, udostępniając podstawowe metody zarządzania danymi oraz konwertując przetwarzane dane na ciągi znaków JSON i odwrotnie. Jako jedyny bezpośrednio korzysta z metod oferowanych przez Ionic Storage.
* PersistencyService
* Z StorageWrappera korzysta explicite PersistencyService, wykorzystujący jego metody w kontekście konkretnych obiektów. Zawiera logikę opartą o interfejs Entity, implementowany przez encje Book i Page. Odwołuje się do danych serwisu Indexer, celem identyfikacji i lokalizacji obiektów zapisanych w bazie.
* Indexer
* Poza encjami Book i Page przechowywana jest również tablica obiektów Index określająca zależności między encjami. Stąd stan początkowy Indexera jest wczytywany z bazy przez PersistencyService na początku działania aplikacji. Informacje o zmianach są sygnalizowane przez serwisy zarządzania stanem aplikacji. Odpowiadając na nie, serwis Indexer zapewnia zgodność tablicy indeksów ze stanem rzeczywistym biblioteki użytkownika. W pewnym stopniu emuluje on możliwości bazy relacyjnej.



Serwisy warstwy dostępu do danych.

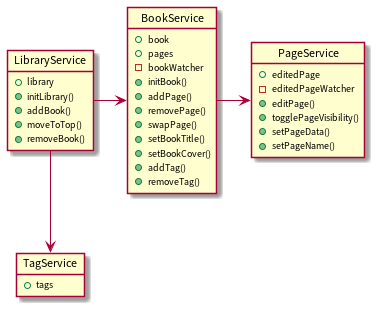
## Warstwa logiki biznesowej

### Serwisy zarządzające stanem aplikacji

Warstwę logiki biznesowej stanowią serwisy zarządzania stanem aplikacji. Wszystkie opierają się na bibliotece Pinia. Umożliwia to funkcjonowanie niezależne od komponentów Vue budujących interfejs aplikacji, udostępniając dla nich globalny stan, do którego mogą się odwoływać.

Aby rozdzielić zadania tej warstwy, serwisy zostały podzielone według domen:

* LibraryService — odpowiada za zarządzanie biblioteką zeszytów użytkownika.
* TagService — udostępnia zbiór tagów utworzonych w ramach biblioteki użytkownika, umożliwiając efektywną kategoryzację zasobów.
* BookService — odpowiada za wprowadzanie zmian w ramach wybranego zeszytu: jego danych oraz zawartości.
* PageService — odpowiada za wprowadzanie zmian w ramach stron wybranej książki.



Serwisy warstwy logiki biznesowej.

Serwisy zarządzania stanem komunikują się z serwisami odpowiadającymi za persystencję danych. Umożliwia to odczyt i zapis danych w pamięci trwałej. Odczytane obiekty, dostarczane przez serwisy persystencji, są traktowane jako stan danego serwisu w ujęciu globalnym — stąd sumarycznie: stan aplikacji. Komponenty Vue mogą korzystać z tych obiektów, modyfikując je za pomocą funkcji udostępnianych przez serwisy. Taki rozdział zadań daje możliwość kontroli akcji podejmowanych przez użytkownika, co pozwala uniknąć nieoczekiwanych zdarzenień podczas działania aplikacji — mogących doprowadzić do błędów. W ramach serwisów obserwowane są zmiany globalnego stanu aplikacji, co pozwala na odzwierciedlenie ich w bazie danych. Ma to miejsce na dwóch płaszczyznach: (1) w przypadku modyfikacji danych obiektu, odpowiedzialność przekazywana jest do PersistencyService; (2) w przypadku modyfikacji przynależności obiektu — jako akcji tworzenia i usuwania zeszytów oraz stron, persystencja realizowana jest w koordynacji z serwisem Indexer. Przyjęty model działania zapewnia w ten sposób spójność danych między warstwami aplikacji.

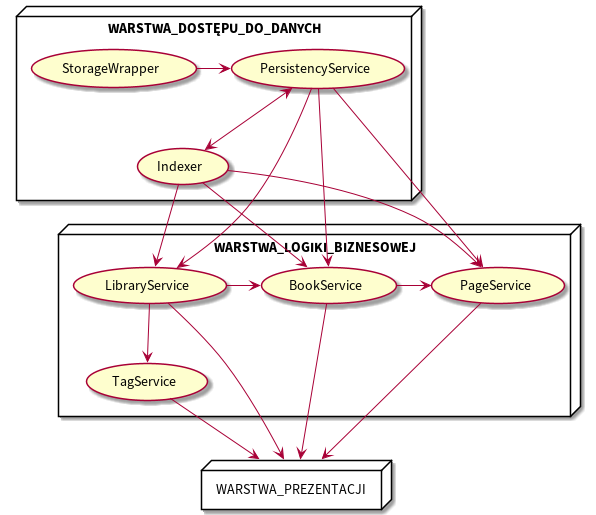


Diagram przedstawiający komunikację warstwy logiki biznesowej z warstwą dostępu do danycha.

## Warstwa prezentacji

### Projekt interfejsu

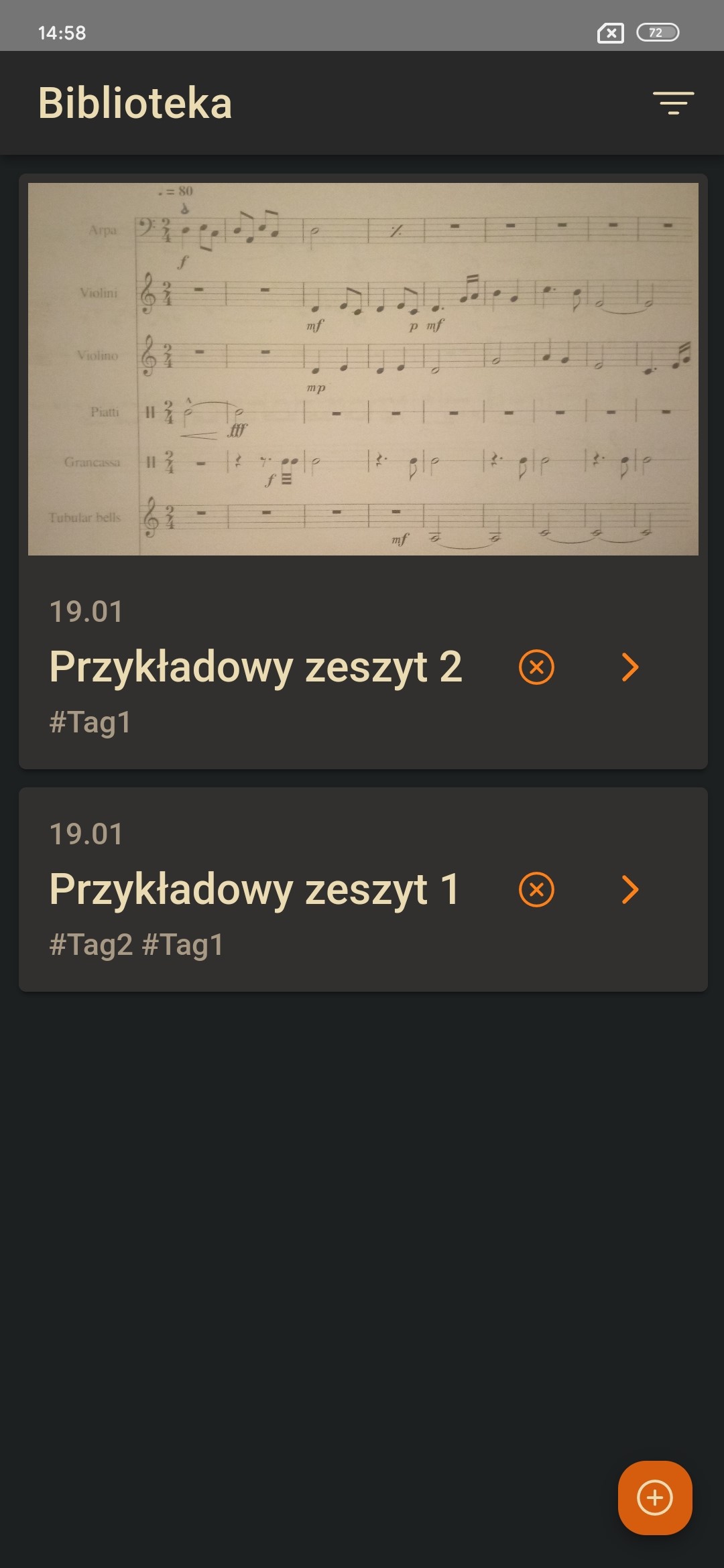
Współgrając z dwupoziomową strukturą modelu danych, projekt interfejsu obejmuje dwa widoki: widok biblioteki oraz widok zeszytu. Są one wspierane przez komponenty główne w postaci oferowanych przez framework Ionic modali oraz menu, rozszerzających funkcjonalności aplikacji. Czerpiąc inspirację ze współczesnych, popularnych koncepcji projektowania interfejsów opartych o modułowość, wizualizacja tworzonych przez użytkownika notatek realizowana jest przez komponenty kart (tu: oferowanych przez framework Ionic). Podobne rozwiązania stosowane są widoczne w przytoczonych w części teoretycznej aplikacjach — m.in. Notion. Karty szczególnie dobrze sprawdzają się enkapsulując zawartość niewielkich rozmiarów. Ich zastosowanie wizualnie oddziela od siebie poszczególne moduły. Z tych powodów przyjęto je jako kluczowe elementy kompozycji aplikacji — nadające kształt zeszytom oraz stronom.

Do edycji składu lub zawartości modułów reprezentowanych przez karty wykorzystano komponenty wysuwanych od dołu ekranu modali. Pozwalają one śledzić proces edycji na bieżąco — zajmując jedynie część ekranu, nie przysłaniają edytowanych kart. Polepsza to również odczucie responsywności aplikacji. Zastosowano również menu, jako komponent dodatkowy, stanowiący przestrzeń edycji danych wybranego zeszytu. Proces ten zwykle nie odnosi się do innych elementów kolekcji, dlatego może być w ten sposób odseparowany od kontekstu.

<diagram przedstawiający drzewo komponentów>

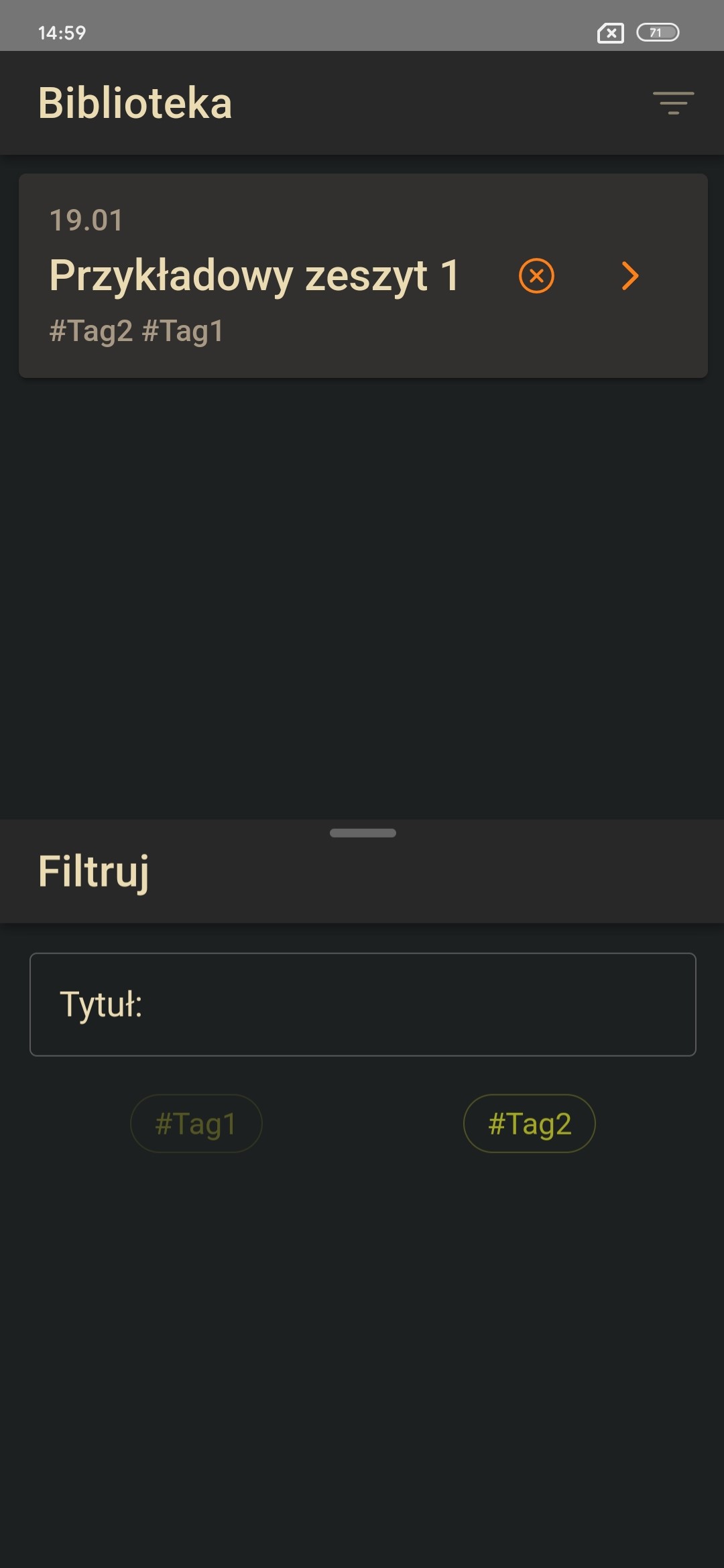
### Implementacja widoków i komponentów głównych

Widok biblioteki umożliwia wizualizację kolekcji zeszytów użytkownika. Na kartach przedstawiających zeszyty widoczny jest tytuł, data modyfikacji oraz przypisane tagi. Jeżeli użytkownik doda do zeszytu okładkę, ułatwiającą identyfikację zeszytu w kolekcji — zostanie tu wyświetlona. Widok biblioteki umożliwia podstawowo trzy akcje: (1) Dodawanie zeszytów, realizowane przez przycisk w dolnym rogu ekranu; (2) Usuwanie zeszytów, po kliknięciu ikony usunięcia na określonej karcie; (3) Otwieranie zeszytów — przejście do widoku zeszytu, po kliknięciu dowolnego miejsca karty (symbol strzałki ma jedynie znaczenie ideowe).



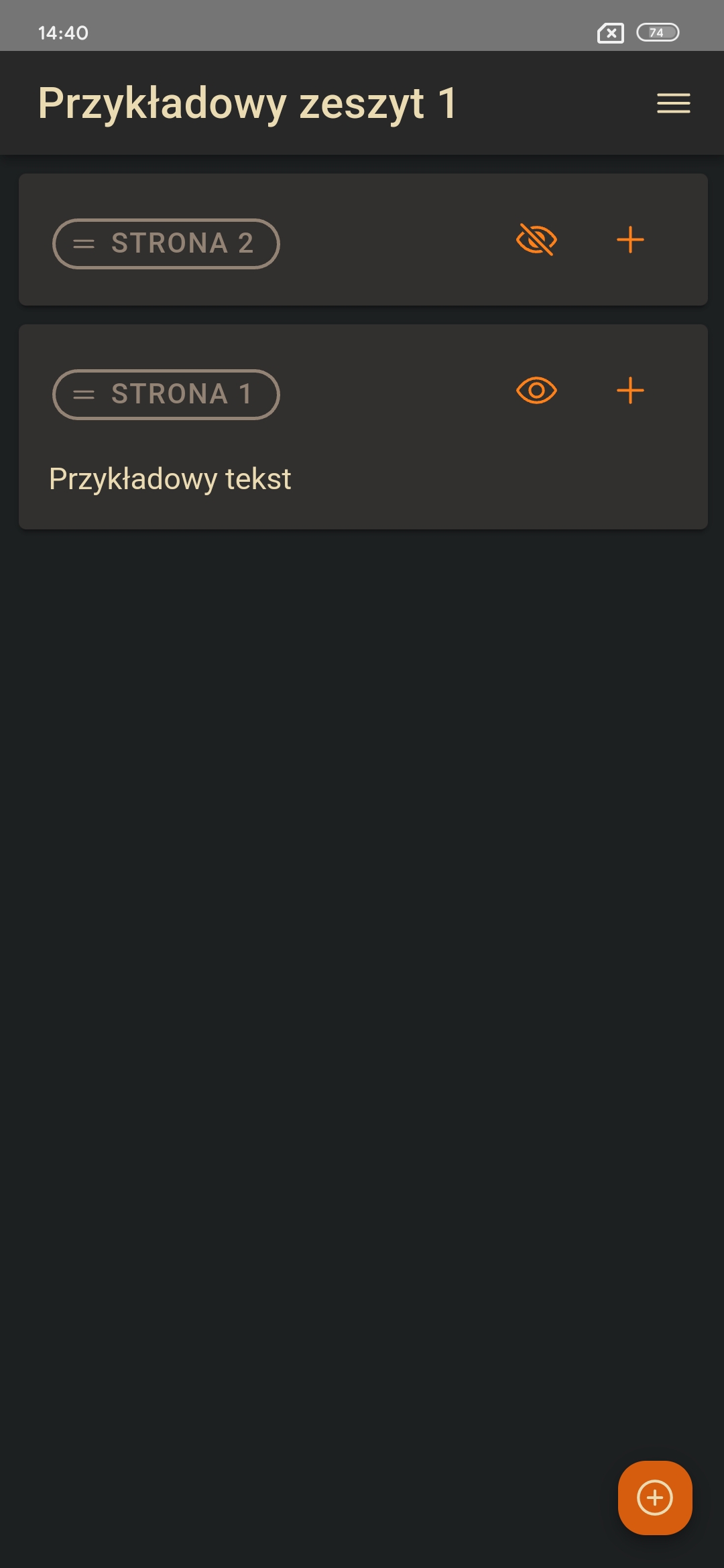
Widok biblioteki.

Możliwość filtracji zeszytów realizowana jest przez modal, otwierany po kliknięciu górnego paska widoku biblioteki. Użytkownik może szukać zeszytów po ich tytułach oraz po słowach kluczowych. Wyszukiwanie po tytule jest case-insensitive.



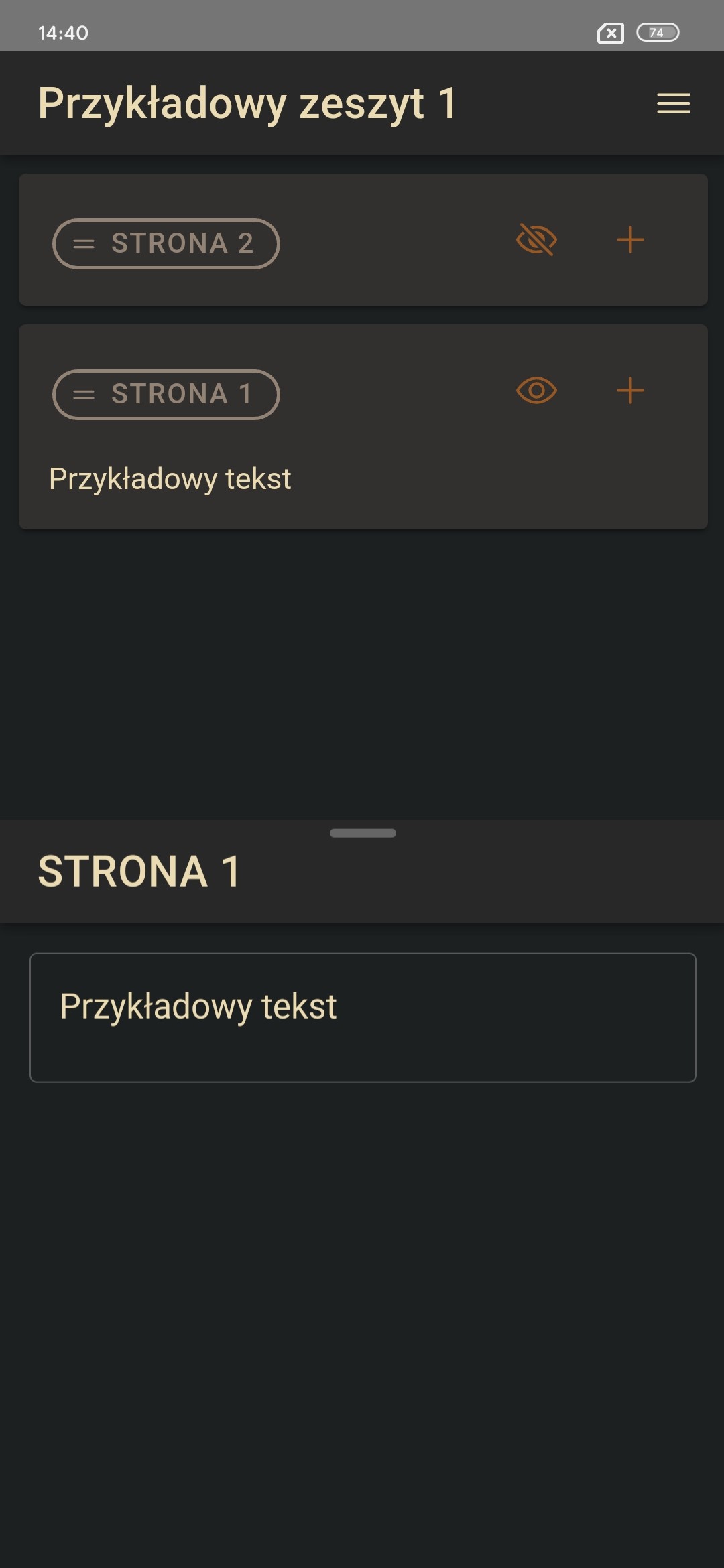
Modal filtrowania.

Widok zeszytu umożliwia wizualizację jego zawartości. Przycisk w rogu ekranu pozwala dodać stronę do zeszytu. Po dodaniu jest ona reprezentowana przez komponent karty. Zawartość strony może zostać ukryta, przy użyciu przycisku z symbolem oka. Przytrzymanie przycisku powoduje trwałe usunięcie strony, sygnalizowane wibracją urządzenia. Strona zawiera także przycisk umożliwiający jej przemieszczenie względem pozostałych stron. W tym miejscu wyświetla się także nazwa strony, jeżeli została nadana przez użytkownika.



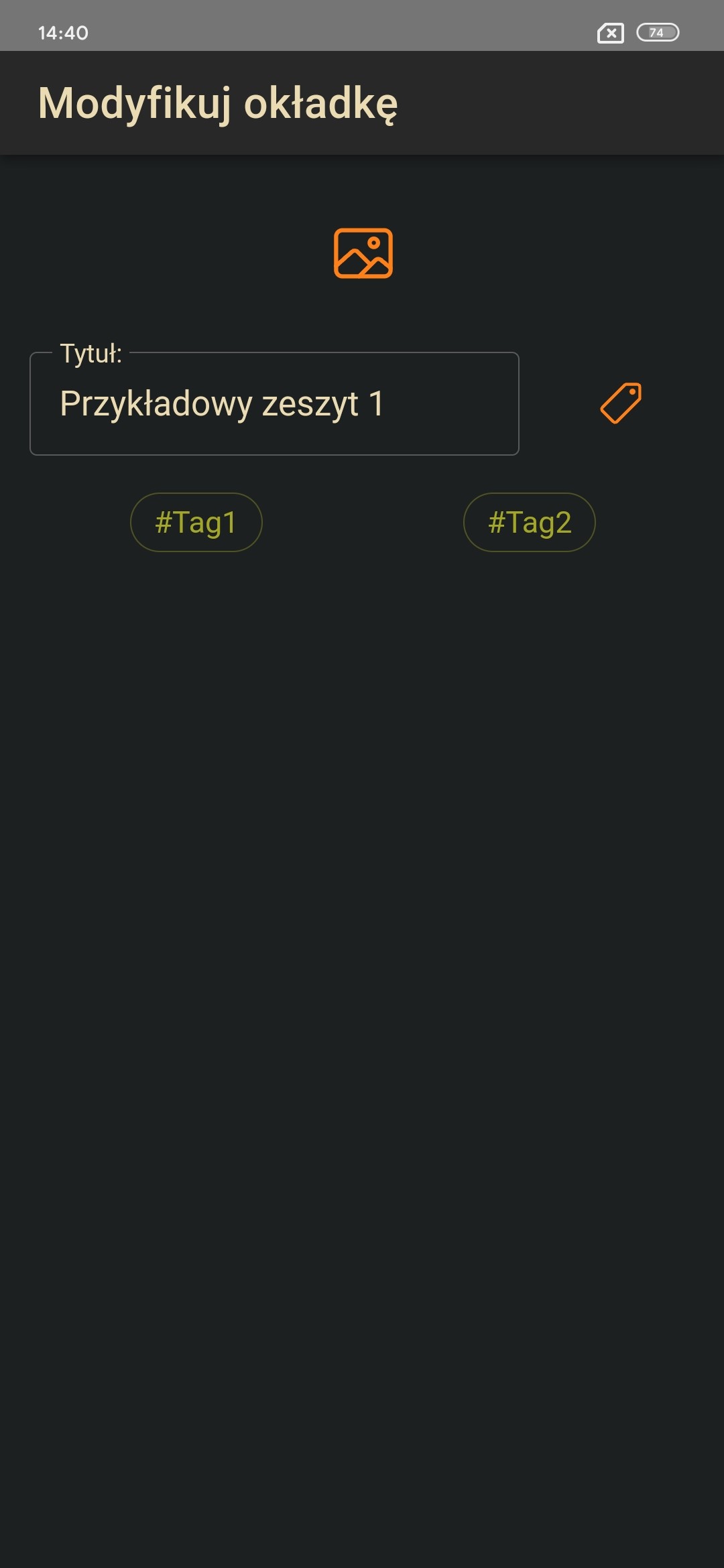
Widok zeszytu.

Możliwość edycji strony realizowana jest przez modal, otwierany po kliknięciu ikony plusa. Zawartość modalu zależy od typu edytowanej strony, zawierając stosowny interfejs edycji zawartości. Każdej stronie można tu również przypisać nazwę.



Modal edycji strony.

Modyfikacja danych zeszytu umożliwiana jest przez menu, otwierane przy kliknięciu górnego paska widoku zeszytu. Przycisk z ikoną zdjęcia umożliwia dodanie okładki; kliknięcie symbolu zakładki pozwala przypisać słowo kluczowe do zeszytu. Można w tym miejscu również zmienić tytuł zeszytu.

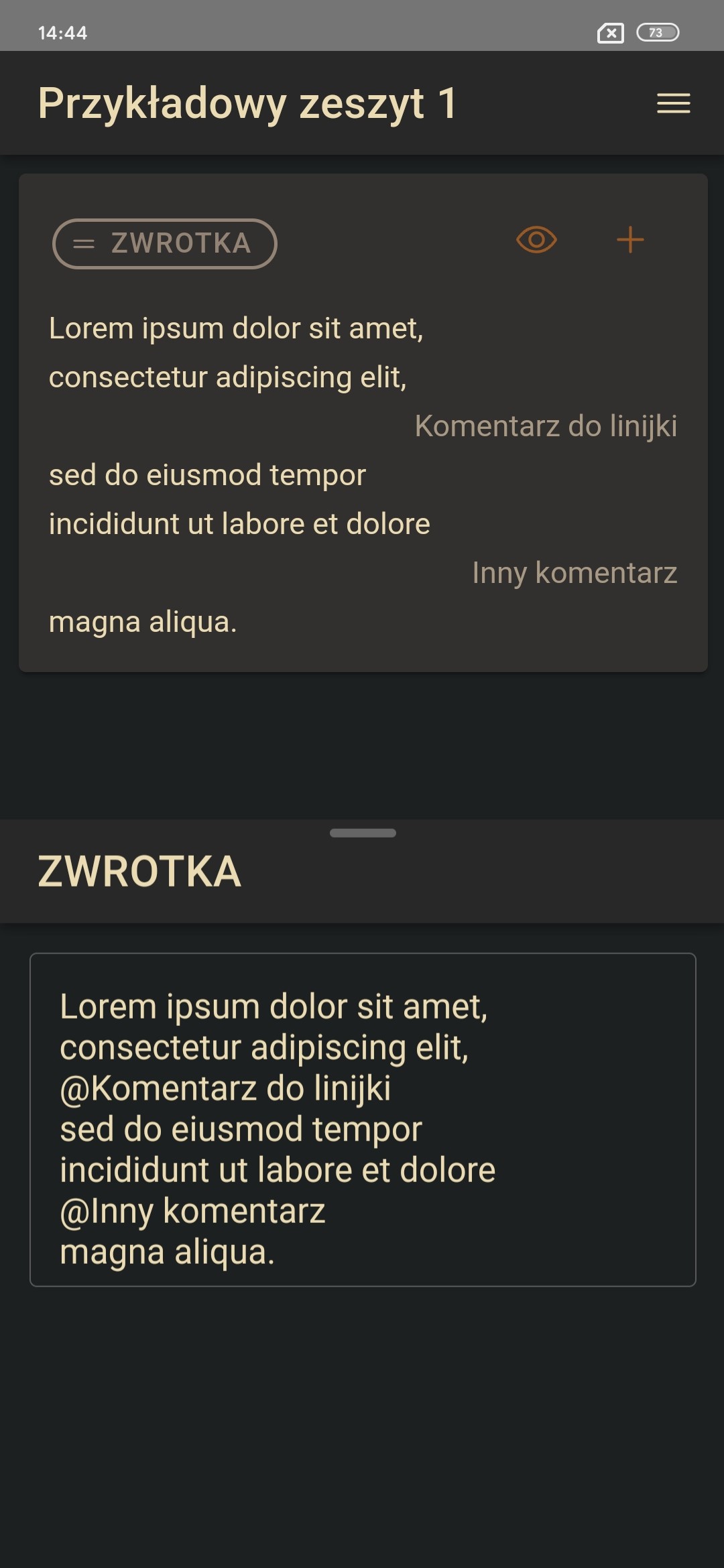


Menu edycji danych zeszytu.

### Implementacja funkcjonalności wspierających aranżację

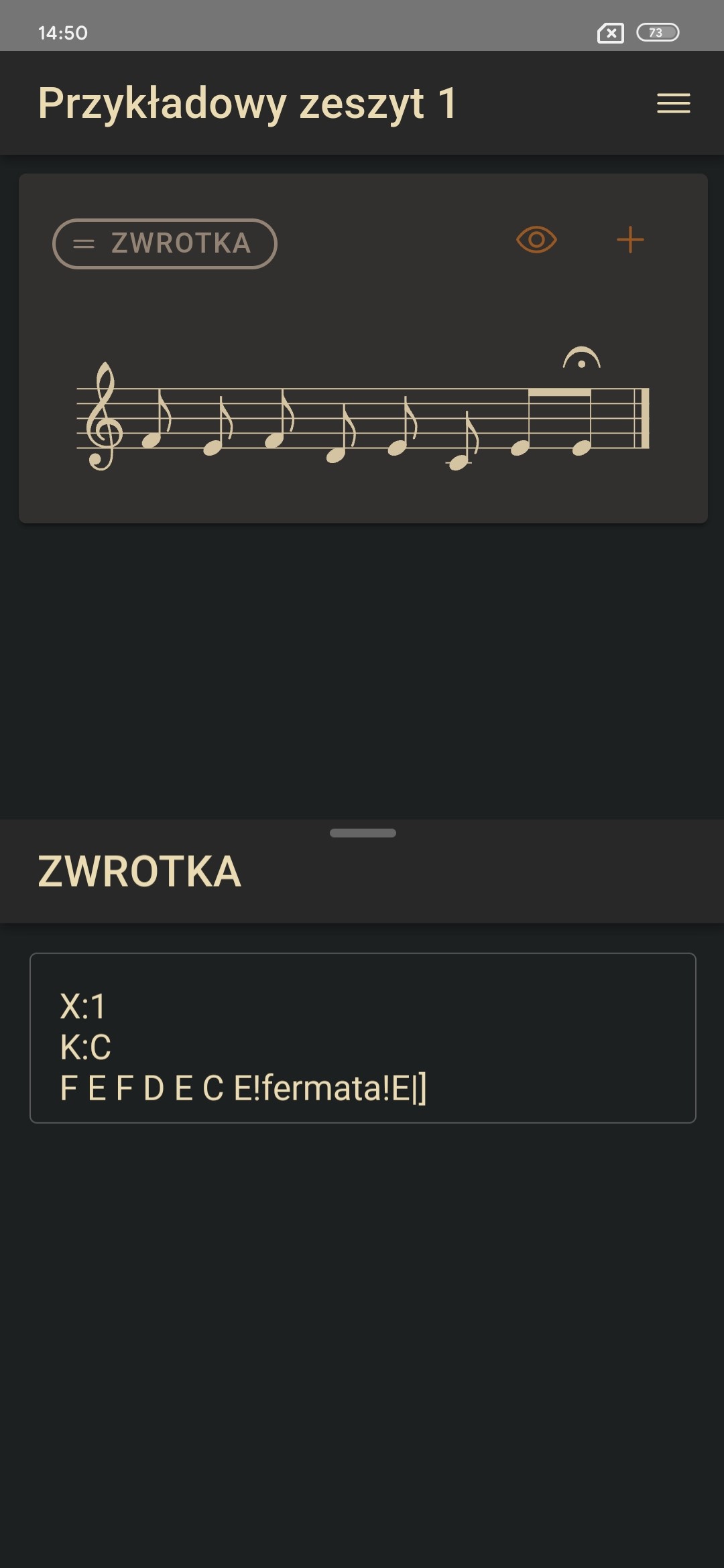
Przedstawione wcześniej komponenty stanowią “środowisko”, które umożliwiaja implementację konkretnych funkcjonalności wspierających proces aranżacji muzycznej. Realizowane są one przez podział stron na cztery — zdefiniowane w wymaganiach aplikacji — typy. Charakteryzują się one odmiennym rodzajem wizualizacji oraz zróżnicowanym procesem edycji. Aby to umożliwić, komponenty główne dopasowują dynamicznie komponenty odpowiadające za wizualizację i edycję stron w zależności od ich typu. Dynamiczne dopasowanie jest realizowane dzięki zastosowaniu component-is oferowanego przez Vue.

Komponenty wizualizacji i edycji tekstu obsługują notatki tekstowe, umożliwiając dodawanie subtelnych (jako: nieodwracających uwagi) komentarzy. Podczas edycji użytkownik zapisuje notatkę w polu tekstowym. Linijka tekstu poprzedzona symbolem “@” traktowana jest jako komentarz. Aby to ułatwić, układ klawiatury przy edycji treści takiej strony ustawiony jest na tryb *e-mail*. Komentarz wyświetlany jest przy prawej krawędzi strony, mniej wyrazistą czcionką.



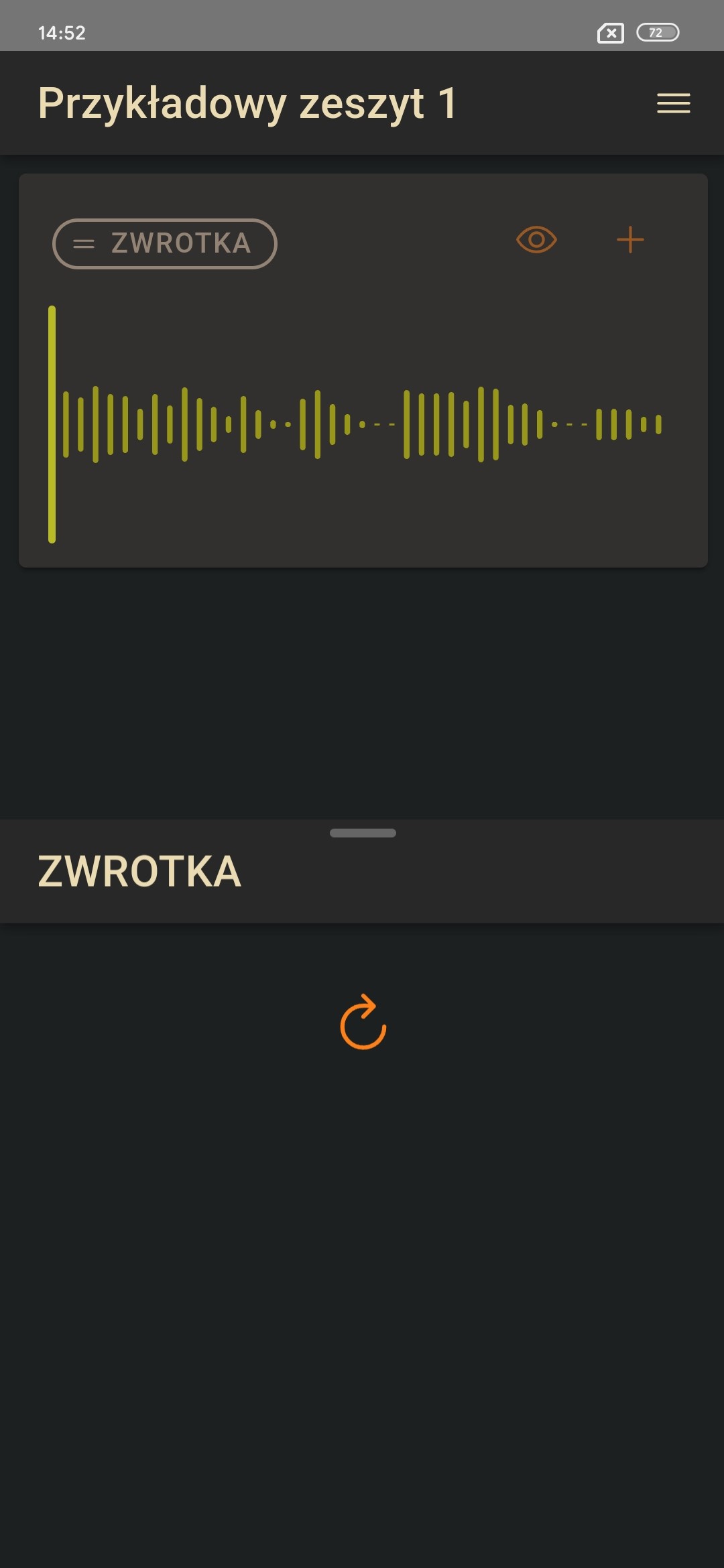
Komponenty strony o typie tekstowym.

Wizualizacja zapisu nutowego realizowana jest przy użyciu biblioteki ABC.js. W ramach komponentu umożliwiającego edycję użytkownik znów zapisuje notatkę w polu tekstowym, lecz jest ona renderowana jako zapis nutowy w komponencie strony. Pomocna jest tu forma komponentu modala, jako umożliwiającego śledzenie zmian zapisu nutowego podczas jego edycji. Z uwagi na przystępną zasadę działania biblioteki ABC.js — za sprawą stosowania nazw literowych do określania nut, implementacja edytorów nut nie jest wymagana. Każda strona o tym typie jest domyślnie inicjalizowana wartościami renderującymi kluczowe elementy notacji muzycznej. Jednocześnie należy zwrócić uwagę, iż użytkownik potrzebujący szerszych możliwości edytora nutowego, powinien zaznajomić się z dodatkowymi funkcjami oferowanymi przez ABC.js.



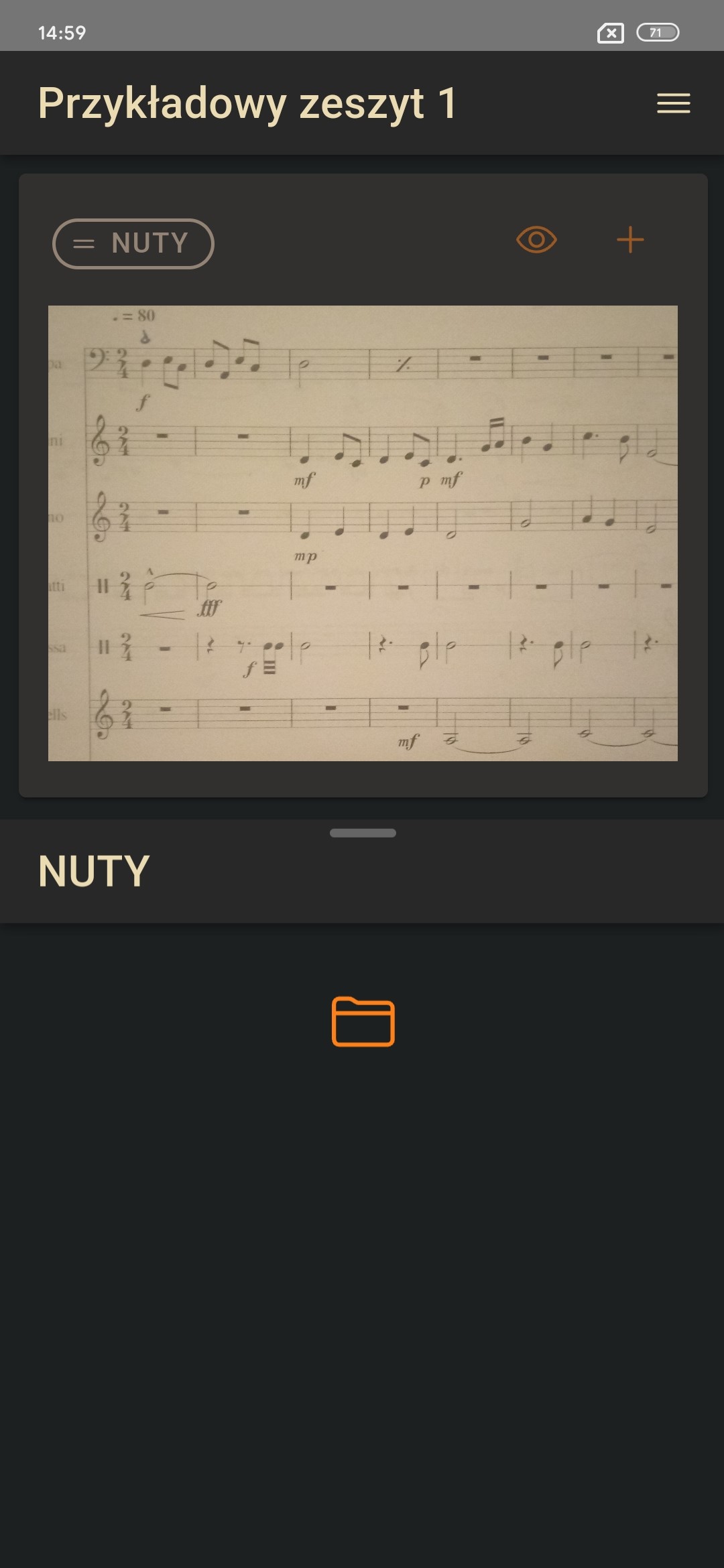
Komponenty strony o typie nutowym.

Komponent edycji notatek dźwiękowych wykorzystuje do działania wtyczkę capacitor-voice-recorder. Pozwala ona nagrać dźwięk, korzystając z mikrofonu urządzenia. Po nagraniu użytkonik może ponownie edytować zawartość strony, zastępując wcześniejszy zapis dźwiękowy nowym nagraniem. Wizualizacja fali dźwiękowej, realizowana przez bibliotekę Wavesurfer.js. Umożliwia orientację w jej przebiegu oraz odtworzenie nagrania od wybranego momentu.



Komponenty strony zawierającym nagranie audio.

Możliwość użycia zdjęcia jako notatki realizowana jest przez wtyczkę capacitor-file-picker, pozwalającej wybrać plik (tu: zdjęcie) z pamięci urządzenia. Wybrane zdjęcie jest wyświetlane jako zawartość strony. Edycja strony polega na możliwości zmiany wybranego zdjęcia.



Komponenty strony zawierającej zdjęcie.

## Testowanie aplikacji

# Analiza otrzymanych wyników i wnioski