Spis treści

Słownik pojęć **3**

Wstęp **4**

Cel i zakres pracy **5**

1. Studium w zakresie problematyki procesu aranżacji muzycznej **6**

1.1. Definicja problematyki procesu aranżacji 6

1.2. Rys historyczny muzyki klasycznej 6

1.2.1. Rozwój technik procesu twórczego muzyki 6

1.2.2. Notacja muzyczna 8

1.2.3. Podsumowanie 9

1.3. Badanie rynku oprogramowania wspierającego pracę muzyków 10

1.3.1. Programy komputerowe 10

1.3.2. Aplikacje mobilne 11

1.3.3. Podsumowanie 11

1.4. Rozwiązania oferowane przez inne aplikacje niepowiązane z procesem aranżacji 11

2. Założenia projektowe **14**

2.1. Identyfikacja aktorów 14

2.2. Poglądowe scenariusze wspierania procesu aranżacji 14

2.3. Wymagania 14

2.3.1. Wymagania jakościowe 14

2.3.2. Wymagania funkcjonalne 16

2.4. Wykaz użytych technologii 17

Część praktyczna **19**

3. Projekt aplikacji wspomagającej aranżacje muzyczne **19**

3.1. Modele danych 19

3.2. Diagramy przepływu 20

3.3. Architektura aplikacji 22

3.4. Warstwa dostępu do danych 25

3.5. Warstwa logiki biznesowej 26

3.6. Warstwa prezentacji 27

3.6.1. Projekt interfejsu 27

3.6.2. Implementacja funkcjonalności wspierających aranżację 32

3.7. Continous Integration i Continous Delivery 35

4. Analiza otrzymanych wyników i wnioski **37**

Bibliografia **39**

Spis rysunków **40**

Słownik pojęć

Celem przybliżenia konceptualnego projektu aplikacji, wykorzystano szereg pojęć, odnoszących się do poszczególnych jej elementów (w nawiasach umieszczone zostały ich angielskie odpowiedniki użyte w kodzie aplikacji):

1. *strona* (Page) — pojedyncza notatka w ramach zbioru
2. *zeszyt* (Book) — zbiór notatek dotyczących jednego utworu, od „zeszytu do nut” (eng. „Music Book”)
3. *biblioteka* (Library) — kolekcja zbiorów notatek użytkownika
4. *warsztat* (Workshop) - ekran edycji notatek

Wstęp

Współcześne rozwiązania programistyczne wspierają działanie wielu dziedzin życia. Udział aplikacji mobilnych wśród innych rodzajów oprogramowania staje się coraz bardziej znaczący, wraz ze wzrostem możliwości telefonów komórkowych. Atut mobilności jest wysoce ceniony, szczególnie w nieprzewidywalnych sytuacjach życia codziennego — zapewniając dostępność, a co za tym idzie — wygodę, niezależną od okoliczności.

Jednocześnie jest wiele dziedzin pracy profesjonalnej, w których zdecydowanie dominują aplikacje na komputery osobiste. Na potrzeby muzyków dostępne są obszerne ekosystemy takich aplikacji, których celem jest wspomaganie procesu powstawania utworów. Dziedzina ta — jako dziedzina sztuki — często charakteryzuje się jednak dużą nieprzewidywalnością. Procesy kreatywne artystów mogą przybierać nieuporządkowane struktury, przynoszące mnogość koncepcji i planów. Wykorzystywane przez muzyków narzędzia programistyczne zdają się często charakteryzować nastawieniem na pracę z koncepcją już *stworzoną i określoną*. Przypadek procesu aranżacji — opracowań utworów — można w tym świetle uznać za szczególny, jako ogniwo pomiędzy kompozycją a interpretacją utworu. Wymagana jest organizacja i dyscyplina, by szkice koncepcji melodii przerodziły się w kompletne dzieło muzyczne. Tkwi stąd potencjał w projekcie aplikacji — jako mobilnego narzędzia pozwalającego uporządkować i określić poszczególne produkty procesu kreatywnego, nadając koncepcji wyrazisty kształt.

Cel i zakres pracy

Celem pracy jest stworzenie aplikacji mobilnej, która umożliwi zapis i organizację koncepcji utworów muzycznych podczas procesu aranżacji. Aplikacja ma stanowić narzędzie o prostej obsłudze, oferujące użytkownikowi dowolność w wyborze sposobu notacji muzycznej utworu. Jej zadaniem jest „cyfrowa emulacja zeszytu do nut" jako rodzaju *muzycznego notesu* oraz rozszerzenie jego możliwości: nie ograniczając się jedynie do zapisu nutowego i tekstowego, lecz wykorzystując również multimedia — jako nośniki informacji o dźwięku. Unikatowość projektu opiera się o ukierunkowanie funkcjonalności aplikacji na wspomaganie części procesów pomijanych przez współczesnie dostępne oprogramowanie.

1. Studium w zakresie problematyki procesu aranżacji muzycznej

1.1. Definicja problematyki procesu aranżacji

Aranżacją w ujęciu muzycznym jest „opracowanie utworu muzycznego w nowym stylu bez zasadniczych zmian linii melodycznej" [31]. Stoi on z tego względu konceptualnie pomiędzy procesami kompozycji oraz interpretacji utworu i w efekcie współdzieli częściowo problematykę obu tych etapów, jako proces techniczny oraz twórczy. Jakość aranżacji zależy od znajomości teorii muzyki, technik kompozytorskich, jak również doświadczenia w pracy z muzyką. Podczas tego procesu, muzyk nadaje utworowi swoisty kształt. Jest to efekt dynamicznej i zmiennej pracy kreatywnej, podczas której wyłania się wiele muzycznych koncepcji, wymagających uporządkowania i selekcji [32].

1.2. Rys historyczny muzyki klasycznej

Analiza historii muzyki na przestrzeni epok odsłania charakter procesu aranżacji. W kontekście omawianego zagadnienia szczególnie istotna jest ewolucja technik twórczych, jak również samej notacji muzycznej. Ukazują one zróżnicowanie podejść twórczych i sposobów pracy z muzyką.

1.2.1. Rozwój technik procesu twórczego muzyki

Rozwój technik wykorzystywanych przez kompozytorów i aranżerów tworzy krajobraz muzyczny następujących po sobie epok. Widoczne jest zróżnicowanie popularnych tendencji wśród twórców muzyki w Europie.

1. **Średniowiecze**:

   Proces twórczy muzyków epoki średniowiecza jest głęboko osadzony w liturgicznym kontekście religijnym. Znaczącą rolę ogdrywa chorał gregoriański jako podstawa tworzonych utworów. Kompozytorzy tego okresu opierają się na zawartych w nim melodiach, wykorzystując techniki — jak na przykład organum, melizmaty oraz schematy rytmiczne — urozmajcając ich brzmienie. Organum polega na dodawaniu nowych dźwięków do

melodii chorału, celem tworzenia wielowarstwowych melodii; melizmaty stosowa są tu jako muzyczne „ozdobniki”. Kluczową rolę w utworze odgrywa jego tekst [22].

2. **Renesans**:

   Odkryciem przełomowym epoki renesansu jest polifonia — nakładanie na siebie równoważnych głosów o odmiennych liniach melodycznych. Szczególną popularnością cieszy się technika imitacji — naśladownictwa. Stosowane są bardziej skomplikowane rytmy, na przykład — opierające się na trójdzielności — oraz nowe skale muzyczne [36].

3. **Barok**:

   Twórczość muzyków epoki baroku charakteryzuje się rozbudowaną ornamentacją. Popularne staje się basso continuo — jako metoda prowadzenia akompaniamentu — oraz technika kontrapunktu, w której kontrastujące melodie tworzą złożone struktury dźwiękowe. Kontynuowane są również eksperymenty z harmonią i dynamiką utworów [9].

4. **Klasycyzm**:

   W okresie klasycyzmu muzyczny proces twórczy cechuje kontrastująca z barokiem precyzja i klarowność. Stosowane są ścisłe formy — jak na przykład forma sonatowa — określające strukturę, tempo, charakter i układ poszczególnych części utworów. Wykorzystywane są powtarzalne motywy melodyczne; widoczne jest dążenie do równowagi między poszczególnymi elementami składowymi utworów [22].

5. **Romantyzm**:

   W podobnym do epok wcześniejszych kontraście epoka romantyzmu przynosi duże oswobodzenie wyrazowe muzyki. Kluczowym elementem motywującym twórczość jest introspekcja, stąd utwory tego okresu nasycone są indywidualizmem i wyrazistością emocjonalną. Wciąż stosowane są pewne gatunki stanowiące „wzorce”, jak sonata czy fantazja, lecz nie są one już tak ściśle określone, jak w klasycyzmie. Pojawiają się stąd zróżnicowane eksperymenty kompozytorskie z nietypową harmonią, dynamicznymi kontrastami i rozbudowanymi frazami melodycznymi [9].

6. **XX wiek**:

   W XX wieku obserwowane jest szczególne bogactwo i różnorodność podejść do kompozycji. Widoczne jest zróżnicowanie nurtów — między innymi impresjonizmu, ekspresjo-

nizmu, neoklasycyzmu. Muzyka staje się „produktem” licznych koncepcji i metod tworzenia — jak aleatoryzm, opierający się o losowość lub serializm, kierowany przetworzeniami matematycznymi; a także idei — można tu przytoczyć minimalizm. Epoka ta przynosi jeszcze jeden przełom — komputery i ich zastosowanie w muzyce, jako narzędzia twórcze, instrumenty oraz nośniki danych [37].

1.2.2. Notacja muzyczna

Na przestrzeni epok kluczowym nośnikiem pozwalającym na utrwalenie melodii jest notacja. Ona również zmienia się i ewoluuje, dostosowując się do rozwoju muzyki oraz potrzeb artystów. Na przestrzeni epok zauważalne jest stąd zróżnicowanie konceptualne zapisu melodii.

1. **Starożytność**:

   Najstarsza forma notacji muzycznej — oparta o zapis klinowy — pochodzi z Mezopotamii (1400 rok p.n.e.). Podobnie późniejsze odkrycia na terenie Grecji — również opierają się o alfabet oraz symbole określające długość dźwięku [36].

2. **Średniowiecze**:

   Pojawia się notacja dedykowana ściśle muzyce — neumy. Określenie to odnosi się do kilku rodzajów notacji, z których pierwsza wywodzi się z Imperium Bizantyjskiego, jednak pod tym pojęciem rozumie się przede wszystkim europejską notację wykorzystywaną początkowo w chorale gregoriańskim — do określania wysokości dźwięków oraz ich czasu trwania. Epoka średniowiecza przynosi również specyficzny rodzaj nazewnictwa dźwięków — solmizację [22].

3. **Renesans**:

   Powstają podwaliny współczesnej ogólnej notacji nutowej: zaczynają być stosowane nuty oraz symbole — jak na przykład klucze. Na przestrzeni kolejnych epok zapis jest rozwijany, umożliwiając bardzo dokładne określanie zamierzonego przez twórcę przebiegu melodycznego. Jednocześnie pojawiają się tabulatury — zapis ukierunkowany pod konkretny instrument, dostosowany do jego charakterystyki — dla lutni, organów itp. [36].

4. **Barok**:

Zaczyna być stosowany bas cyfrowany. Jest to notacja numeryczna określające harmonię akompaniamentu utworów. Rola ta przypada zwykle konkretnym instrumentom, najczęściej — klawesynowi. Zasada działania tego rodzaju notacji opiera się o akordy i relacje interwałowe pomiędzy poszczególnymi dźwiękami.

5. **Klasycyzm i romantyzm**:

   Kolejne epoki przynoszą dalszy rozwój standaryzowanego zapisu nutowego, celem oznaczenia ścisłej artykulacji, dynamiki i agogiki dla wykonawców utworów [22].

6. **XX wiek**:

   Muzycy zaczynają eksperymentować z zapisem muzycznym, zupełnie dostosowując go do osobistych potrzeb, tworząc warstwy abstrakcji, własne „paradygmaty", odwzorowujące innowacyjność tworzonej muzyki [37].

7. **Współczesność**:

   Współcześnie korzysta się z wielu form zapisu muzycznego. Kluczową rolę odgrywa zapis nutowy, lecz niektórzy artyści wykorzystują także inne formy notacji, podobnie jak miało to miejsce w XX wieku. Jednocześnie w pewnych okolicznościach wciąż wykorzystywane są starsze metody zapisu — sięgające nawet do neum. Wielką popularnością cieszą się również tabulatury, zapis akordowy oraz wizualizacje komputerowe — z uwagi ta ich przystępność [34].

1.2.3. Podsumowanie

Analiza rysu historycznego ukazuje szeroką gamę wykorzystywanych technik i sposobów notacji muzycznej. Dostępność edukacji muzycznej oraz instrumentów przekłada się na duże zróżnicowanie sposobów pracy artystów. W następstwie XX—wiecznych tendencji kompozytorskich oraz przemian społeczno—kulturowych osiągnięta zostaje całkowita dowolność w sposobach pracy z muzyką.

1.3. Badanie rynku oprogramowania wspierającego pracę muzyków

1.3.1. Programy komputerowe

Praca twórcza muzyków od XX wieku jest również wspierana przez programy komputerowe. Można zaobserwować dynamikę postępu w tej dziedzinie, na przykładzach następujących po sobie nowatorskich rozwiązań programistycznych na przestrzeni lat:

1. **1950 - początki zastosowania komputerów w muzyce**:

   W 1951 roku inżynier komputerowy Max Mathews tworzy pierwszy program komputerowy do generowania dźwięku. Program, działając na komputerze IBM 704, otwiera drzwi do eksperymentów z syntezą dźwięku przy użyciu komputerów. W późniejszych latach powstaje ”MUSIC I”— tego samego twórcy — umożliwiający tworzenie muzyki. Można określić ten moment jako początek muzyki komputerowej [29].

2. **1980 - MIDI (Musical Instrument Digital Interface) i edycja dźwięku**:

   Wprowadzenie standardu MIDI umożliwia komunikację między różnymi instrumentami muzycznymi a komputerami, co znacząco ułatwia produkcję i edycję muzyki przy użyciu oprogramowania. Pojawiają się też programy takie jak SoundEdit i Sound Forge, umożliwiające edycję dźwięku. Zyskują one popularność w dziedzinie produkcji muzycznej [7].

3. **1990 - DAW (Digital Audio Workstation), syntezatory i samplowanie**:

   Powstanie programów DAW, takich jak Pro Tools, Cubase, Logic Pro, Ableton Live i FL Studio, umożliwia profesjonalistom tworzenie, edycję i produkcję muzyki na komputerach osobistych. DAW stają się kluczowym narzędziem w przemyśle muzycznym. Pojawia się również oprogramowanie umożliwiające emulację syntezatorów analogowych oraz samplowanie dźwięków [7].

4. **2000 — Rozszerzanie zastosowań programów komputerowych**:

   Oprogramowanie takie jak Kontakt, Omnisphere czy Massive pozwala na tworzenie wirtualnych instrumentów i efektów dźwiękowych, rozszerzając możliwości produkcji muzycznej. Powstają programy automatyzujące procesy jak Ludwig 3.0 — tworzący automatyczne aranżacje, Transcribe — transkrybujący dźwięk na notację muzyczną, a także szereg innych projektów wspomagających muzyków w procesie twórczym [23].

1.3.2. Aplikacje mobilne

Współcześnie, przy wzroście możliwości smartfonów należałoby oczekiwać również wzrostu ich zastosowań w dziedzinie muzyki. Rynek jednak oferuje dość nieliczne rozwiązania — często nastawione na wykonywanie pojedycznych, prostych czynności — istnieją grupy aplikacji umożliwiających tworzenie zapisu nutowego lub służących za metronomy [13]. Znaczący potencjał aplikacji mobilnych jest za to widoczny na przykładzie GarageBand, będącego pełnym programem DAW, pozwalającym na tworzenie muzyki przy użyciu smartfona. Umożliwia korzystanie z syntezatorów, nagrywanie, edycję dźwięku, jak również możliwość nakładania efektów dźwiękowych. Mimo to, w przypadku tego rodzaju pracy, preferowane są z rozwiązania desktopowe, które — choćby z uwagi na rozmiar ekranu — oferują większy komfort [14].

1.3.3. Podsumowanie

Analiza programów i aplikacji stanowiących zestaw narzędzi muzyków na przestrzeni lat oraz współcześnie odsłania niezapełnioną niszę: wciąż brakuje rozwiązań — tak wśród aplikacji mobilnych, jak i programów komputerowych — które mogłoby wspomóc pracę twórczą w fazach powstawania koncepcji, podczas aranżacji utworów muzycznych. Dotychczasowe rozwiązania pozwalają zapisywać ułożone utwory oraz poddawać je edycji; także umożliwiają generowanie aranżacji bądź całych utworów. Jednocześnie elementy procesu aranżacji muzycznej, wiążące się z mnogością pomysłów i zmian — nieczęsto są wspierane przez rozwiązania programistyczne.

1.4. Rozwiązania oferowane przez inne aplikacje niepowiązane z procesem aranżacji

Aby odpowiedzieć na problem omawiany w pracy, konieczna jest analiza oprogramowania przynoszącego rozwiązania problemów o podobnym charakterze. Pod tym kątem, najbliższym przykładem jest oprogramowanie umożliwiające tworzenie notatek. Można zaobserwować rozwój wielu przełomowych — a współcześnie kluczowych — funkcjonalności oferowanych przez aplikacje z tej kategorii na przestrzeni ostatnich dwudziestu lat:

1. **2000 - EverNote**:

Evernote, założone przez Stepana Pachikova, jest jednym z najbardziej rozpoznawalnych narzędzi do tworzenia notatek. Aplikacja pozwala tworzyć notatki w różnych formatach, jako tekst, obrazy, dźwięki, oraz umożliwia synchronizację ich między różnymi urządzeniami [8].

2. **2003 - Microsoft OneNote**:

   Microsoft OneNote (pierwornie OneNote 2003), jest początkowo narzędziem do tworzenia notatek w systemie Windows. W kolejnych edycjach umożliwia użytkownikom innowacyjne organizowanie notatek w ramach „tablicy" [28].

3. **2008 - Evernote jako aplikacja mobilna**:

   Zmiana kierunku rozwoju aplikacji za sprawą wejścia na rynek aplikacji mobilnych doprowadza do wzrostu popularności Evernote, co ukazuje potencjał aplikacji do notowania, będących zawsze „pod ręką" użytkowników. Inne aplikacje również podążają za tym trendem [8].

4. **2013 - Notion**:

   Znaczącą innowację przynosi Notion. Stanowi kolejny krok w ewolucji aplikacji do notowania, jako tzw. personalna baza wiedzy (eng. „Personal Knowledge Base"). Choć sama aplikacja charakteryzuje się stosunkową prostotą, jest jednocześnie to swoistą „piaskownicą danych", dowolnie organizowanych przez użytkownika, przy użyciu notatników, kalendarzy i tabel [25].

5. **2020 - Obsidian, Logseq**:

   Kolejny przełomowy kierunek rozwoju wyznacza aplikacja Obsidian. Charakteryzuje się większą prostotą od Notion, działając w oparciu o pliki markdown, jednak użytkownik może ją dowolnie dostosować do swych potrzeb m.in. za sprawą wtyczek [27]. Podobne założenia realizowane są przez Logseq, oprogramowanie open-source, które opiera się o listy punktowane oraz grafy [21].

Na przykładzie najnowszych aplikacji, widoczna jest tendencja oddawania w ręce użytkowników prostych narzędzi, które mogą być wykorzystane na wiele sposobów — zgodnie z jego zamysłem i stylem pracy. Wyłaniają się stąd również kluczowe założenia współczesnych aplikacji do notowania: mobilność, elastyczność i organizacja. Wymienione aplikacje reprezentują

także różne podejścia do wizualizacji *notatki*. Może być to strona z tekstem (Obsidian), element na liście punktowanej (Logseq), karta w układzie kanban (Notion), element umiejscowiony w dowolnym położeniu na tablicy (Microsoft OneNote) itd. Współczesne trendy wśród aplikacji do notowania odsłaniają również szczególne znaczenie wizualnej || przestrzennej organizacji notatek.

2. Założenia projektowe

2.1. Identyfikacja aktorów

Aktorem korzystającym z zaprojektowanej aplikacji jest jej użytkownik. W założeniu jest to osoba zajmująca się muzyką, szczególnie: pracująca nad tworzeniem i aranżacją utworów. Nie jest tu brany pod uwagę profesjonalizm użytkownika; program ma odpowiedzieć jako podręczny notatnik przede wszystkim na potrzeby muzyków, których proces twórczy nie przebiega w sposób całkowicie usystematyzowany, często jako „efekt chwili".

2.2. Poglądowe scenariusze wspierania procesu aranżacji

Poniżej przedstawiono przykładowe sytuacje problematyczne, jakie mogłaby rozwiązać projektowana aplikacja:

1. Aranżer ma kilka pomysłów na pewien fragment utworu, ale jeszcze nie wie, który z nich wykorzysta: może wszystkie je zapisać jako nagrania oraz opisać słowami, do późniejszej weryfikacji.
2. Muzyk uczestniczy w próbie zespołu, dyrygent proponuje mu alternatywną linię melodyczną: korzystając z aplikacji, muzyk może zapisać ją w formie nutowej.
3. Aranżer pracuje nad utworem, ma kilka fragmentów melodycznych, które zamierza wykorzystać: może dowolnie je zorganizować w aplikacji, ułatwiając ich lokalizację.

2.3. Wymagania

2.3.1. Wymagania jakościowe

Biorąc pod uwagę charakter problemu oraz czerpiąc inspirację z pokrewnych współczesnych rozwiązań, zdecydowano się na rozwiązanie będące aplikacją mobilną. Jest to szczególnie istotne, przez wzgląd na wygodę i dostępność. Telefony komórkowe stanowią współcześnie narzędzie uniwersalne, będące zawsze w zasięgu ręki [13], stąd są najbardziej przystępną platformę dla aplikacji, umożliwiając realizację „podręcznego notatnika".

Kolejną wymaganą cechą aplikacji jest jej niezależność od połączenia sieciowego. O ile rozwiązania sieciowe przynoszą wiele potencjalnie przydatnych funkcjonalności, takich jak współdzielenie danych między urządzeniami, o tyle (pomijając kwestię dodatkowego skomplikowania architektury programu) sama aplikacja nie powinna być od nich zależna, z uwagi na warunki pracy muzyków. Będąc w trasie, podczas pobytu na sali koncertowej, bądź nawet w salach prób — mogą mieć oni ograniczony dostęp do internetu, z uwagi na lokalizację, bądź architekturę budynku [Wymagane źródło bibliograficzne]. Stąd, aby zapewnić niezawodność działania aplikacji bez względu na warunki, wymagana jest *lokalna* praca aplikacji — bez połączenia sieciowego.

W kwestii zasady działania — jako notatnik — aplikacja powinna stanowić swoiste „środowisko", w którym użytkownik decyduje o tym, w jaki sposób je wykorzysta. W ten sposób aplikacja może odpowiedzieć na personalne potrzeby artysty, dostosowując się do jego technik i charakteru pracy.

Aby interfejs aplikacji nie stanowił przeszkody w pracy, lecz współgrał z użytkownikiem, musi cechować się przejrzystością, intuicyjnością oraz prostotą. Przyjmując za ideę minimalizm, jako kierunek projektu interfejsu użytkownika, osiągany jest jednocześnie wysoki stopień interaktywności — zawierając tylko to, co konieczne można pozwolić, by każdy widoczny element umożliwiał określoną funkcjonalność i odpowiadał na akcje użytkownika.

Przedstawione powyżej założenia stanowią podstawę do sformułowania kluczowych wymagań jakościowych aplikacji:

1. Mobilność.
2. Niezawodność.
3. Niezależność od warunków użytkowania.
4. Prostota.
5. Przejrzystość.
6. Intuicyjność.
7. Interaktywność.

2.3.2. Wymagania funkcjonalne

W najszerszym ujęciu aplikacja ma umożliwić użytkownikowi notowanie muzycznych koncepcji aranżacji utworów. Odnosząc się do zróżnicowania notacji muzycznej — aby pokryć możliwie szerokie spektrum metod zapisu muzyki, wyszczególnione zostały formaty:

1. Nutowy — jako podstawowa i uniwersalna forma zapisu dźwięku.
2. Tekstowy — jako medium pozwalające opisywać dźwięk, jak również zapisywać teksty utworów, chwyty gitarowe.
3. Dźwiękowy — jako nagranie, umożliwia przechwycenie i zapis brzmienia.
4. Wizualny — jako zdjęcia, przykładowo nut, ustawienia zespołu podczas prób.

Częstą praktyką wśród muzyków jest również notowanie wskazówek wykonawczych (na przykład artykulacyjnych) na stronach z nutami. Przydatne okazuje się stąd umożliwienie użytkownikowi tworzenie podobnych adnotacji, jako komentarzy przy nutach lub między liniami tekstu.

Możliwość organizacji elementów umożliwia ich wyszukiwanie oraz lokalizację. Biorąc przykład z przytoczonych wcześniej aplikacji do tworzenia notatek, może mieć ona charakter wizualny, opierający się na umiejscowieniu elementu na określonej przestrzeni, w odniesieniu do innych elementów. Także pomocna jest możliwość wizualnej modyfikacji samego elementu — wyróżniając go na tle innych.

Skuteczna organizacja wymaga jednak również systematycznej struktury — poza wymiarem wizualnym — celem umożliwienia wyszukiwania elementów według określonych przez użytkownika grup. Wybrano w tym celu mechanizm tagowania (oznaczania słowami kluczowymi), jako bardziej elastyczny od struktury drzewiastej (realizowanej na przykład jako *system plików i folderów*). Umożliwia on przynależność pojedynczego elementu do wielu grup.

Na podstawie powyższych założeń, można ująć wymagania funkcjonalne aplikacji jako:

1. Zapisywanie i edycja notatek opisujących założenia aranżacyjne utworu jako: nuty, tekst, nagranie, obraz.
2. Opatrywanie komentarzami fragmentów notatek nutowych i tekstowych.

3. Organizacja przestrzenna notatek.
4. Organizacja zbiorów notatek poprzez nadawanie tytułów i tagów.
5. Wizualne wyróżnianie zbiorów notatek przez dołączanie zdjęć.

2.4. Wykaz użytych technologii

Celem zrealizowania projektu wykorzystano następujące kluczowe technologie:

1. Node.js (18.16.1) oraz npm (9.5.1)

   Node.js to środowisko uruchomieniowe JavaScript na silniku V8 [24]. Narzędzie npm (Node Package Manager) jest integralną częścią ekosystemu Node.js, zapewniając mechanizm zarządzania zależnościami oraz dostarczając modułów i pakietów, co znacząco ułatwia proces tworzenia oraz zarządzanie projektem opartym na Node.js [26].

2. TypeScript (5.1.6)

   Język programowania rozwijany przez Microsoft, będący nadzbiorem języka JavaScript. Dzięki systemowi typów TypeScript wspiera programistów w wykrywaniu błędów na etapie kompilacji, co zwiększa niezawodność i utrzymanie kodu w projektach [35]. Z tego względu, jak również z uwagi na jego znajomość przez Autora pracy, wykorzystany jest jako główny język projektu.

3. Ionic Framework (cli — 7.1.5)

   Framework do budowy hybrydowych aplikacji mobilnych, korzystający z języków webowych takich jak HTML, CSS i JavaScript bądź TypeScript. Ionic CLI ułatwia tworzenie, testowanie i wdrażanie aplikacji mobilnych na różne platformy, na podstawie jednego, wspólnego kodu źródłowego [15]. Wybrano tę technologię z uwagi na jej uniwersalność międzyplatformową, oferującą dalsze potencjalne kierunki rozwoju projektu.

4. Capacitor (core — 5.4.0)

   Narzędzie do budowy natywnych aplikacji mobilnych, przy użyciu technologii webowych. Capacitor umożliwia dostęp do funkcji natywnych urządzeń jak na przykład aparatu czy mikrofonu. Współgra on z frameworkiem Ionic, celem dostarczania aplikacji na urządzenia z systemami Android i IOS [4].

5. Vue.js (3.2.45)

   Progresywny framework JavaScript do budowy interfejsów użytkownika. Vue.js umożliwia łatwe tworzenie interaktywnych interfejsów, dzięki deklaratywnemu podejściu do komponentów i reaktywnemu systemowi danych. Tworzony jest z myślą o przystępności i intuicyjności [38] — z tego względu został wybrany do wykorzystania, w połączeniu z Ionic.

6. Pinia (2.1.4)

   Biblioteka do zarządzania stanem globalnym aplikacji. Stanowi następstwo Vuex store — jako dedykowana dla Vue.js. Umożliwia ona również korzystanie z narzędzi deweloperskich pozwalających śledzić zmiany stanu aplikacji [30].

7. Ionic Storage (4.0.0)

   Umożliwia przechowywanie i pobieranie danych w aplikacjach mobilnych, zapewniając prosty interfejs do manipulacji danymi, przy użyciu bazy klucz—wartość. Jest aktywnie wspierany oraz zgodny z Ionic w wersji 7, co nie jest aktualnie oferowane inne biblioteki persystencji [16].

8. Biblioteki JavaScript do obsługi notatek:

   Sortable — dla umożliwienia wygodnej organizacji przestrzennej [33], wavesurfer.js — wizualizujący przebieg nagrania w formie dźwiękowej [39], abcjs — renderujący notację nutową na podstawie tekstu [1].

9. Środowiska i aplikacje programistyczne: JetBrains WebStorm — główne środowisko programistyczne przygotowania projektu [40], GitHub oraz Git — celem realizacji kontroli wersji [11][10], Buddy CI/CD — jako narzędzie do Continous Integration i Continous Delivery, umożliwiające automatyczne budowanie projektu do postaci aplikacji mobilnej [3].

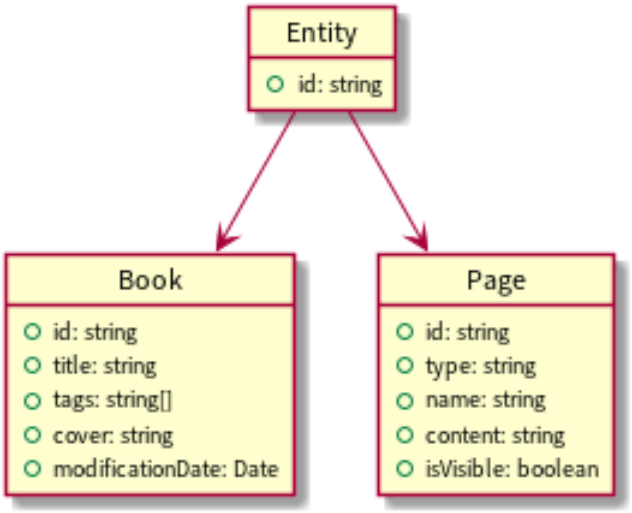
Część praktyczna

3. Projekt aplikacji wspomagającej aranżacje muzyczne

3.1. Modele danych

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.

Pierwszy poziom stanowi koncepcja „zeszytu do nut" (dalej okeślanego pojęciem „zeszyt"), jako zbioru notatek reprezentowany przez encję Book. Idąc dalej tym tokiem, przyjęto określenie „strona" jako encja Page, odnoszące się do poziomu drugiego — pojedynczej notatki użytkownika. Obie encje implementują prosty interfejs Entity, deklarujący pole id, ułatwiając działanie serwisów zarządzających persystencją.



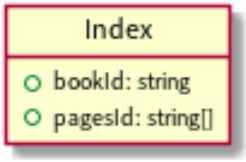
Rys. 1. Modele encji implementujące interfejs Entity.

Model encji Book zawiera pola: title — jako tytuł zeszytu, tags — przechowujące opisujące go słowa kluczowe, cover — będące zapisem base64 [2] zdjęcia, ustawionego jako okładka zeszytu, oraz automatycznie ustawiane modificationDate — wskazujące na datę ostatniej modyfikacji zeszytu. Model encji page zawiera pola: type — określające typ strony, name — za-

wierające jej nazwę, content — przechowujące zawartość oraz isVisible — wyznaczające stan widoczności zawartości strony. Zawartość strony jest zapisywana jako tekst lub w przypadku zawartości multimedialnej — w formacie base64 [2].

Identyfikatory encji są tworzone uniformowo, jako sygnatura czasowa systemu Unix, w momencie utworzenia obiektu. Synchroniczność realizowanych przez aplikację akcji tworzących instacje obiektów gwarantuje unikatowość identyfikatorów występujących w bazie. Po utworzeniu obiektu jego identyfikator nie podlega modyfikacji.

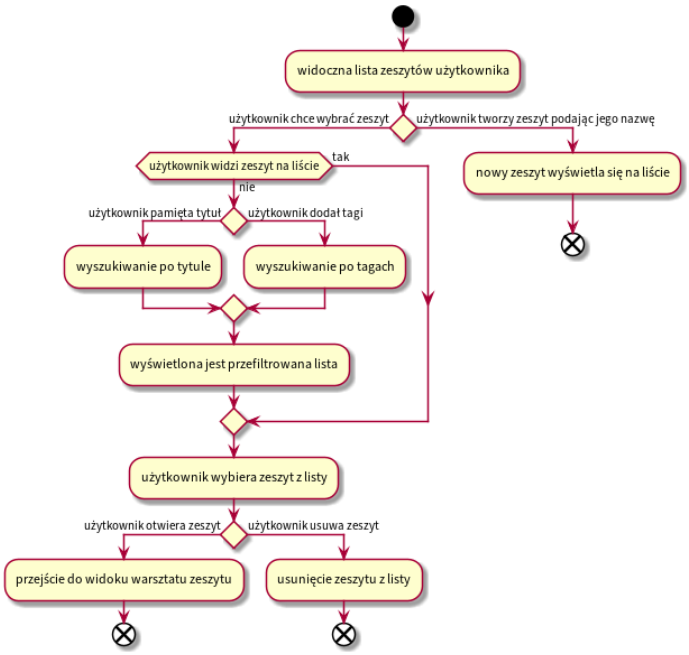
Jako wyznacznik zależności między encjami Book i Page, w bazie przechowywana jest dotatkowo tablica obiektów Index. Z uwagi na jej nadrzędny charakter, nie implementuje ona od interfejsu Entity. Występuje w bazie pojedynczo, pod ustalonym kluczem „INDEX".



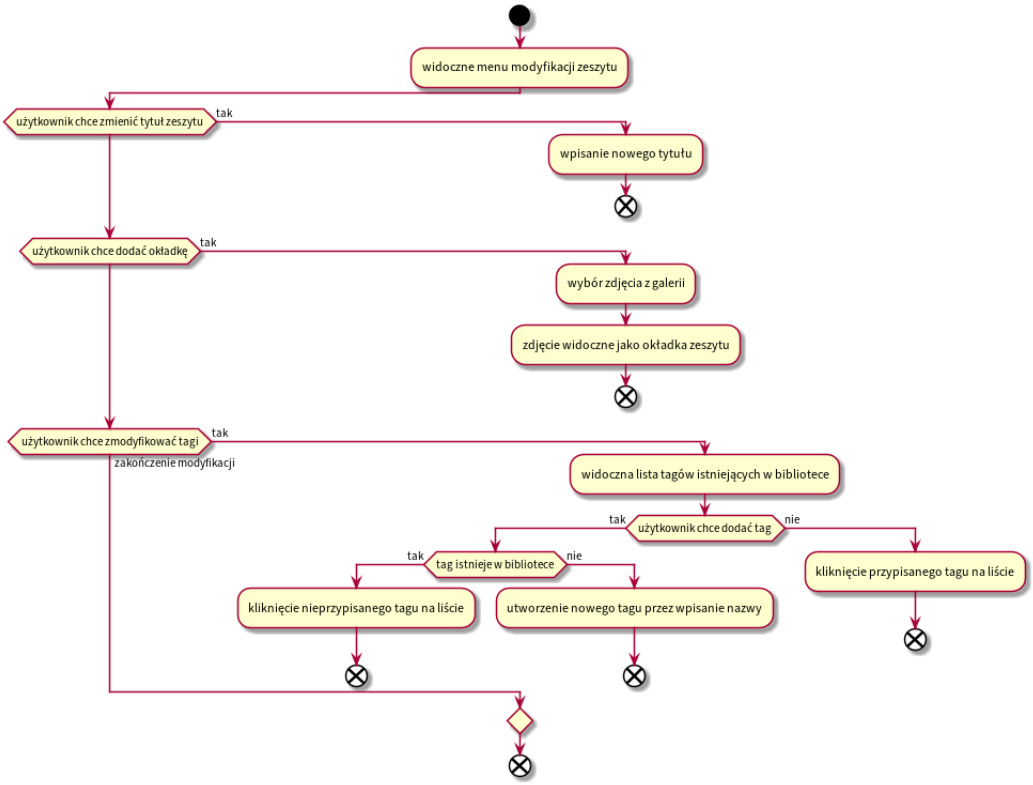
Rys. 2. Model obiektu Index.

3.2. Diagramy przepływu

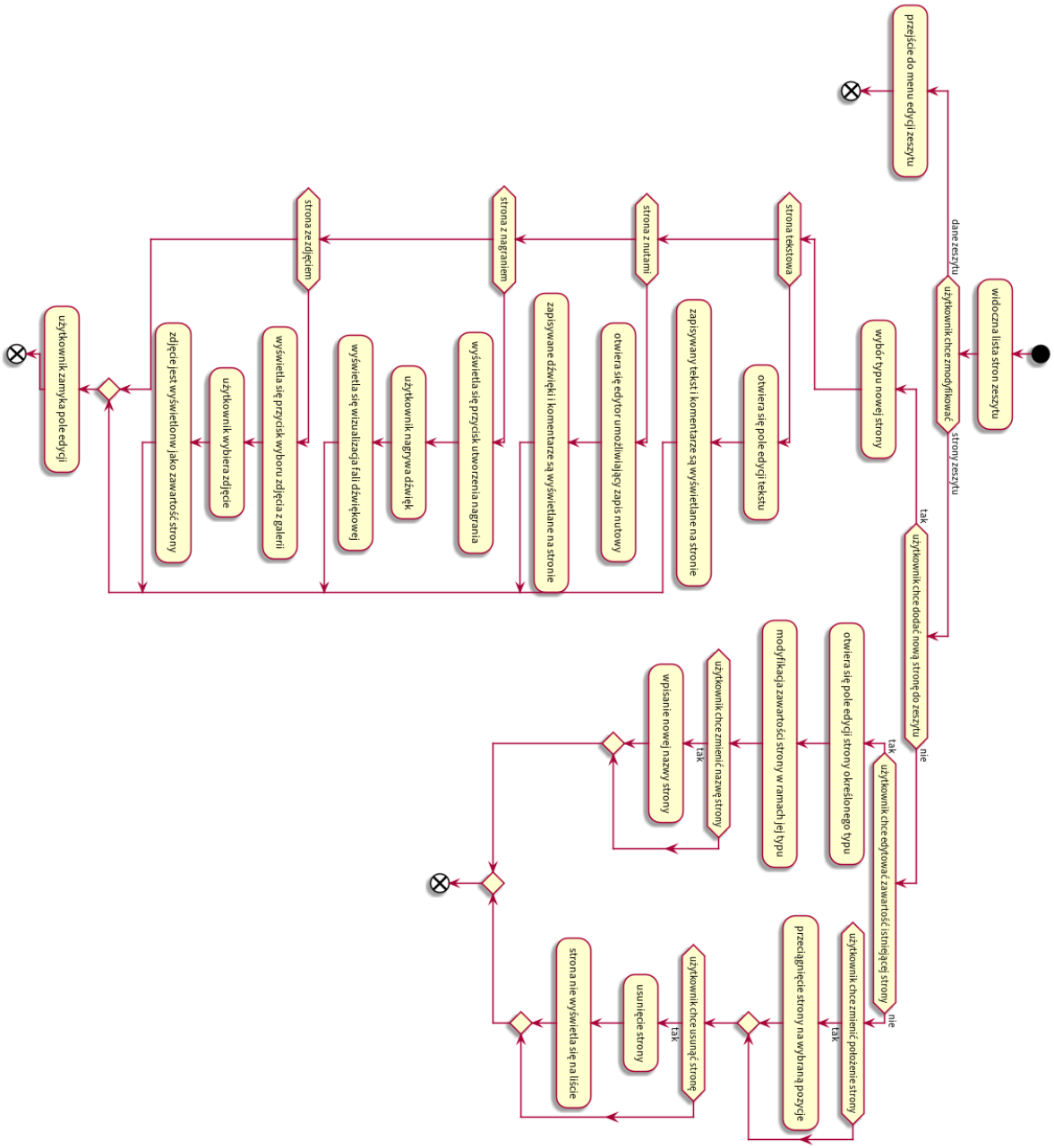
Diagram UX (ang. user flow experience) oddaje założenia dotyczące przebiegu „doświadczeń użytkownika" (ang. user experience || UX) w ramach projektowanej aplikacji. Na podstawie przyjętych konwencji dotyczących nazewnictwa oraz założeń projektowych, można zobrazować planowane działanie aplikacji z punktu widzenia jej użytkownika:



Rys. 3. Diagram przepływu dla widoku biblioteki zeszytów użytkownika.



Rys. 4. Diagram przepływu dla menu edycji zeszytu.



Rys. 5. Diagram przepływu dla widoku warsztatu użytkownika.

3.3. Architektura aplikacji

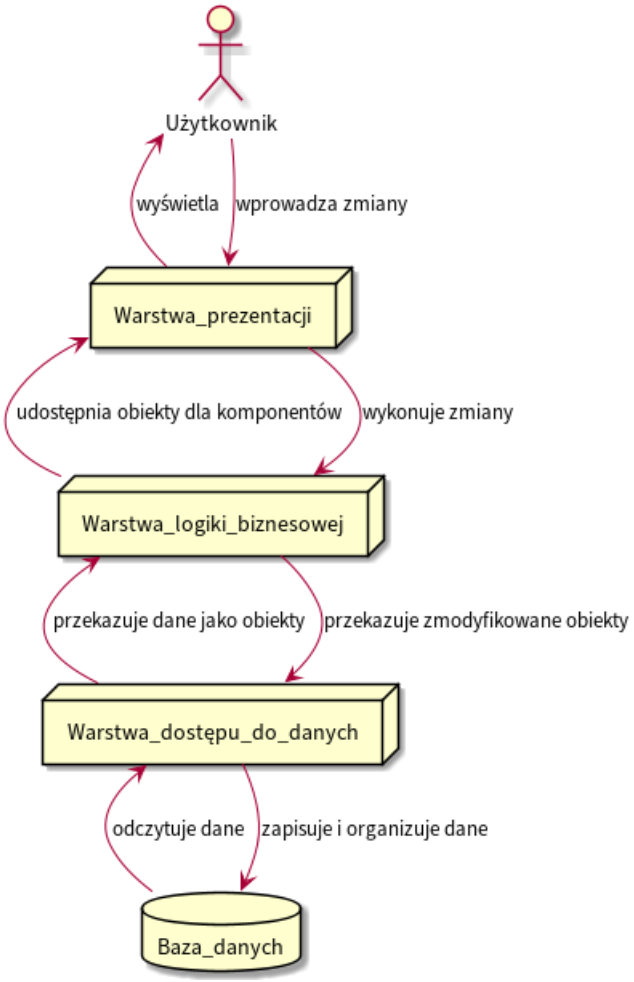
Jako model architektury aplikacji przyjęto architekturę warstwową. Pozwala ona odseparować zadania wykonywane przez poszczególne komponenty aplikacji, ułatwiając utrzymanie kodu. Jest to osiągane poprzez rozgraniczenie zadań realizowanych przez poszczególne warstwy. Architektura warstwowa charakteryzuje się hierarchicznym układem komponentów, co

przekłada się na zwiększenie przejrzystości kodu. W ramach projektu zastosowano trzy warstwy podziału:

1. Warstwa prezentacji — realizowana przez komponenty Vue, stanowiące interfejs użytkownika. Wykonuje podstawowe operacje jak obsługa wtyczek do wizualizacji elementów lub funkcje formatowania tekstu.

2. Warstwa logiki biznesowej — obejmująca szczególnie serwisy zarządzania stanem aplikacji, stanowiące punkt wykonania akcji podejmowanych przez użytkownika. Odnosi się do danych pobieranych z bazy.

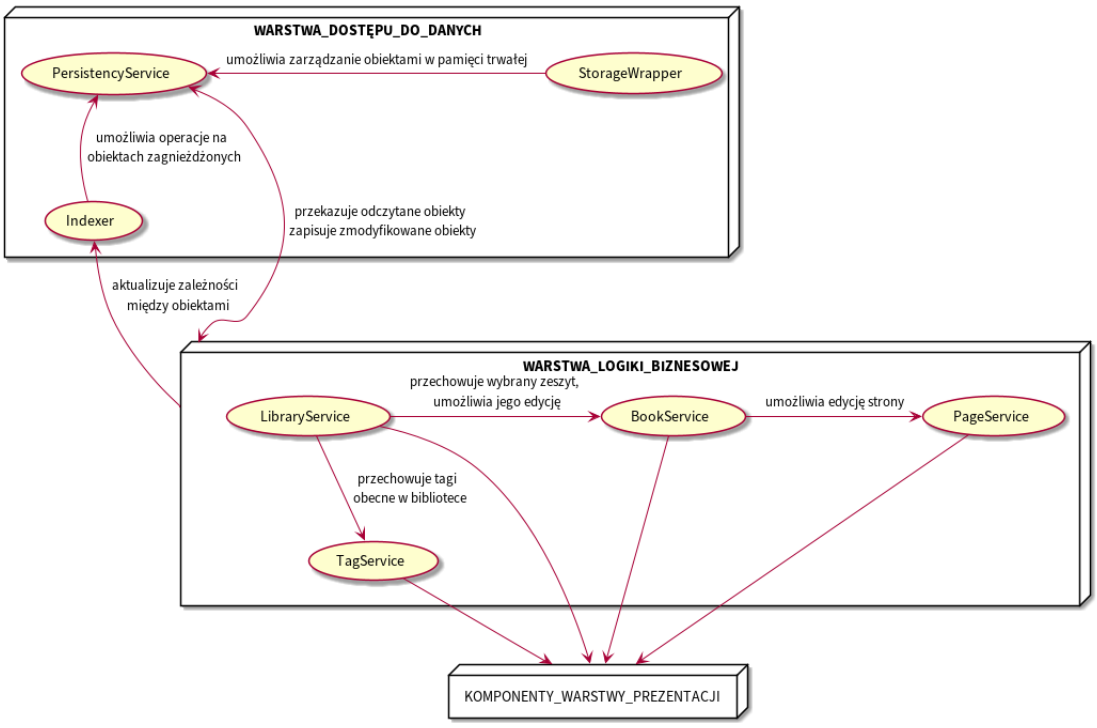
3. Warstwa dostępu do danych — odpowiadająca za komunikacje z bazą danych, w oparciu o model encji. Przekazuje dane modyfikowane w warstwie logiki biznesowej, zapewniając ich trwałość.



Rys. 6. Diagram realizacji architektury warstwowej.

Serwisy warstwy logiki biznesowej komunikują się z serwisami warstwy dostępu do danych. Otrzymują w ten sposób obiekty notatek, zapisane trwale w pamięci urządzenia. Odczytane obiekty są traktowane jako *stan* danego serwisu w ujęciu globalnym — stąd sumarycznie: *stan aplikacji*. Dzięki temu komponenty Vue warstwy prezentacji mogą z nich korzystać — wyświetlając je oraz umożliwiając edycję. Wprowadzanie zmian należy z powrotem do zadań warstwy logiki biznesowej. Obserwacja zmian *stanów* serwisów tej warstwy pozwala odzwierciedlić je w bazie danych.

Ma to miejsce na dwóch płaszczyznach: (1) w przypadku modyfikacji danych obiektu, odpowiedzialność trwałego zapisu zmienionego obiektu przekazywana jest do PersistencyService; (2) w przypadku modyfikacji przynależności obiektu — jako akcji tworzenia i usuwania zeszytów oraz stron — zapis zmian realizowany jest w koordynacji z serwisem Indexer. Taki rozdział zadań daje możliwość kontroli akcji podejmowanych przez użytkownika, co pozwala uniknąć nieoczekiwanych zdarzeń podczas działania aplikacji — mogących doprowadzić do błędów.



Rys. 7. Diagram przedstawiający komunikację między serwisami warstw aplikacji.

3.4. Warstwa dostępu do danych

W aplikacji zastosowano bazę danych typu klucz-wartość Ionic Storage [16]. Aby umożliwić zarządzanie obiektami zagnieżdżonymi — jako: *zeszyt* zawierać może wiele *stron* — przy zachowaniu niezależności operacji na obiektach, zastosowano trzy serwisy:

1. StorageWrapper

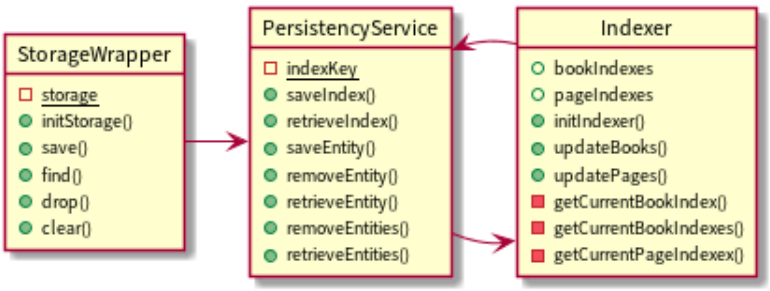
   Z uwagi na charakter kluczy i wartości bazy — jako dwóch ciągów znaków — obiekty zapisywane w bazie są konwertowane do ciągów JSON [18]. StorageWrapper *opakowuje* bibliotekę Ionic Storage, udostępniając podstawowe metody zarządzania danymi oraz konwertując przetwarzane dane na ciągi znaków JSON i odwrotnie. Jako jedyny bezpośrednio korzysta z metod oferowanych przez Ionic Storage [16].

2. PersistencyService

   Z serwisu StorageWrapper korzysta PersistencyService, wykorzystujący jego funkcje w kontekście konkretnych obiektów. Zawiera logikę wykorzystującą interfejs Entity, implementowany przez encje Book i Page. Odwołuje się do danych serwisu Indexer celem identyfikacji i lokalizacji obiektów zapisanych w bazie.

3. Indexer

   Poza encjami Book i Page przechowywana jest również tablica obiektów Index określająca zależności między encjami, za którą odpowiada serwis Indexer. Jego stan początkowy jest wczytywany z bazy przy uruchomieniu aplikacji. Informacje o zmianach są sygnalizowane przez serwisy warstwy logiki biznesowej. Odpowiadając na nie, serwis Indexer zapewnia zgodność tablicy indeksów ze stanem rzeczywistym biblioteki użytkownika. W pewnym stopniu emuluje on możliwości relacyjnej bazy danych.



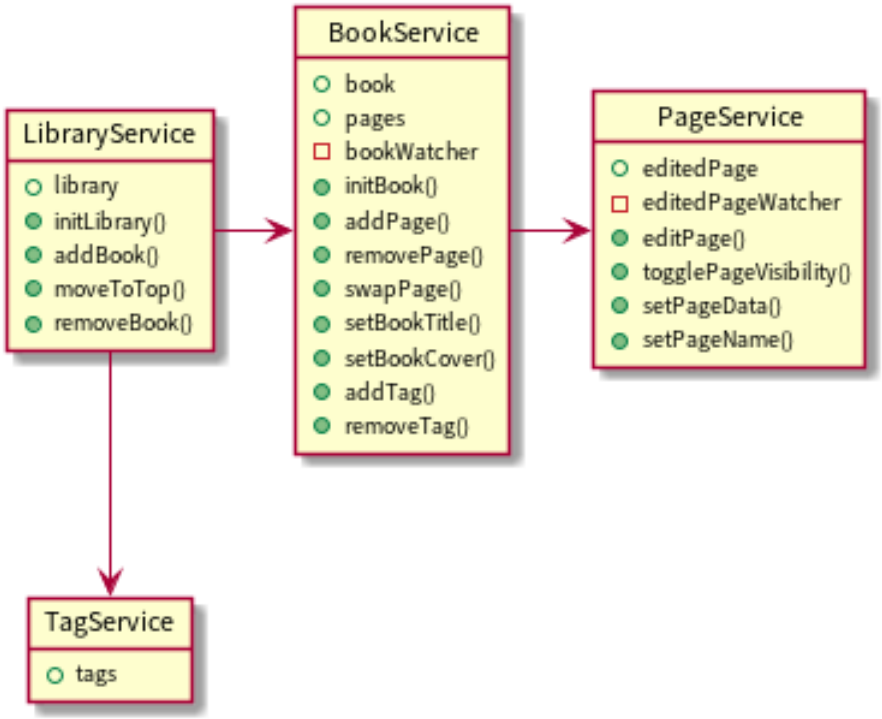
Rys. 8. Serwisy warstwy dostępu do danych.

3.5. Warstwa logiki biznesowej

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:

1. LibraryService — odpowiada za zarządzanie biblioteką zeszytów użytkownika.
2. TagService — udostępnia zbiór tagów utworzonych w ramach biblioteki użytkownika, umożliwiając efektywną kategoryzację zasobów.
3. BookService — odpowiada za wprowadzanie zmian w kontekście zeszytu: jego danych oraz zawartości.
4. PageService — odpowiada za edycję stron wybranego zeszytu.



Rys. 9. Serwisy warstwy logiki biznesowej.

3.6. Warstwa prezentacji

3.6.1. Projekt interfejsu

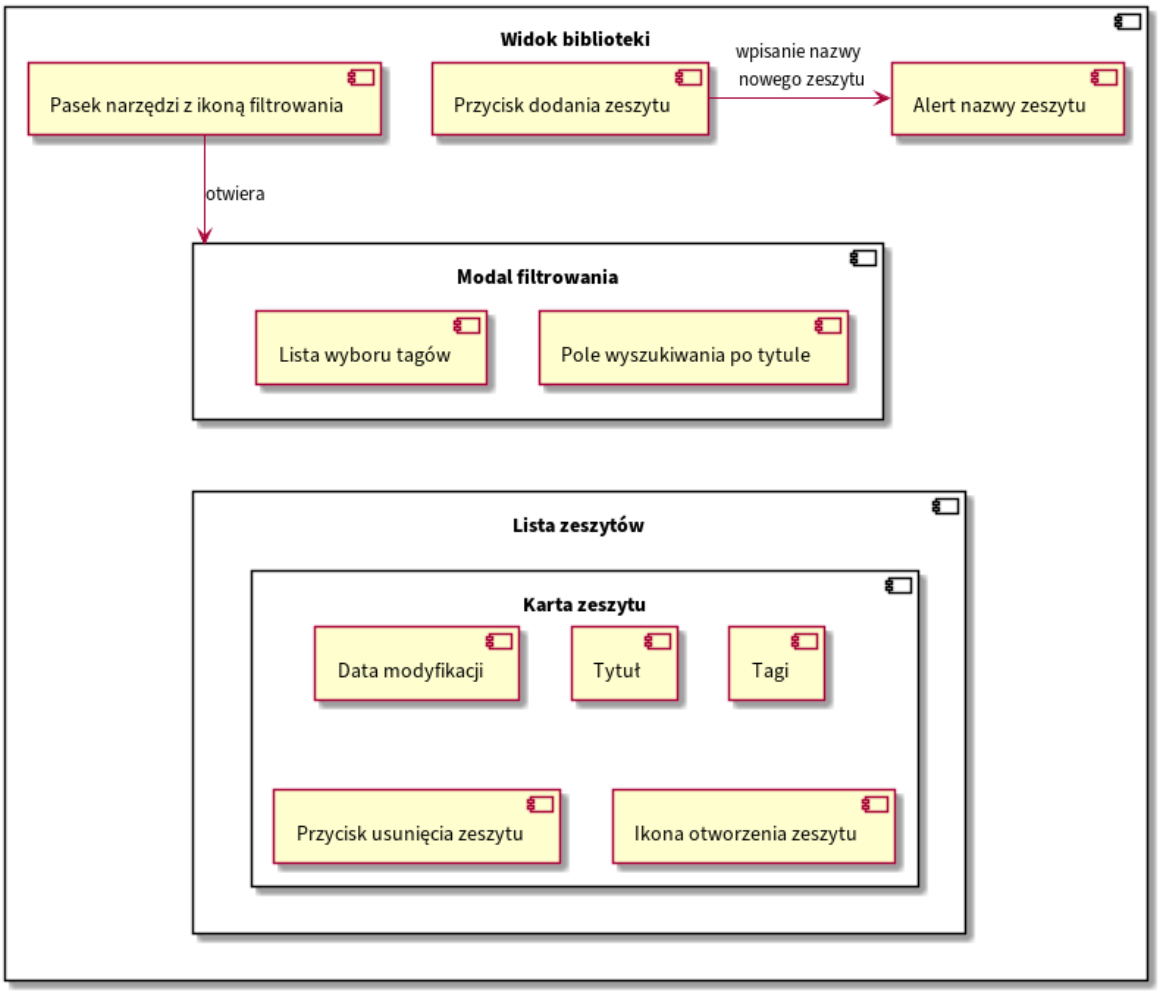
Współgrając z dwupoziomową strukturą modelu danych, projekt interfejsu obejmuje dwa widoki: widok biblioteki i widok warsztatu. Są one wspierane przez komponenty główne w postaci oferowanych przez Ionic Framework modali oraz menu [15], rozszerzających funkcjonalności aplikacji.

Czerpiąc inspirację ze współczesnych, popularnych koncepcji projektowania interfejsów opartych o modułowość, wizualizacja tworzonych przez użytkownika notatek realizowana jest przez komponenty kart (tu: oferowanych przez Ionic Framework [15]). Podobne rozwiązania stosowane są w przytoczonych w części teoretycznej aplikacjach — m.in. w Notion [25]. Karty szczególnie dobrze sprawdzają się enkapsulując zawartości 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.

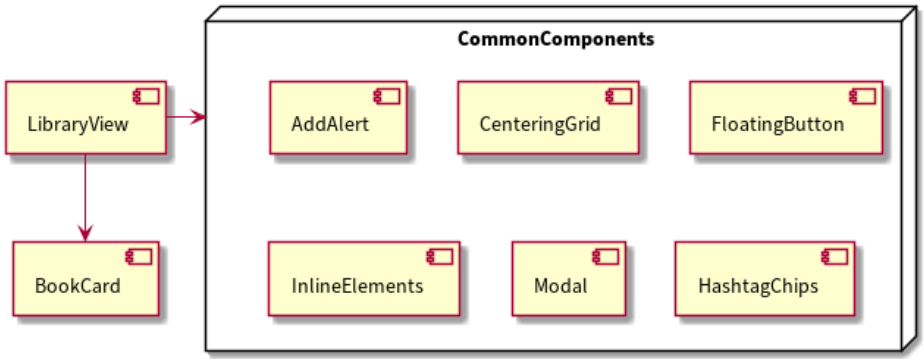
Do edycji składu lub zawartości modułów reprezentowanych przez karty wykorzystano komponenty wysuwanych od dołu ekranu modali [15]. Pozwalają one śledzić proces edycji na bieżąco — zajmując jedynie część ekranu, nie przysłaniają edytowanych kart. Polepsza to również odczucie responsywności aplikacji. Zastosowano również menu [15], 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.

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 [15] na określonej karcie; (3) Otwieranie zeszytów — przejście do widoku zeszytu, po kliknięciu dowolnego miejsca karty (ikona strzałki [15] ma jedynie znaczenie symboliczne).

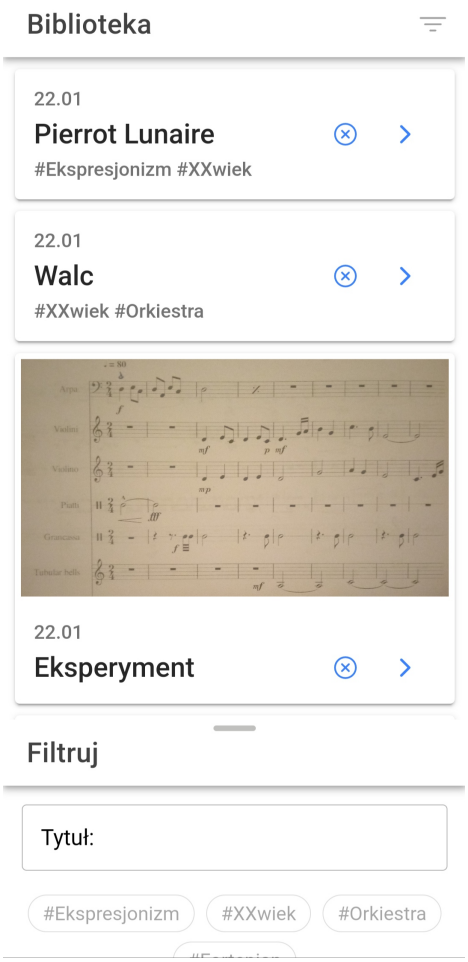
Możliwość filtracji zeszytów realizowana jest przez modal, otwierany po kliknięciu górnego paska widoku biblioteki. Użytkownik może szukać zeszytów po ich tytułach oraz po słowach kluczowych.



Rys. 10. Diagram interfejsu widoku biblioteki.



Rys. 11. Drzewo komponentów Vue realizujących widok biblioteki.



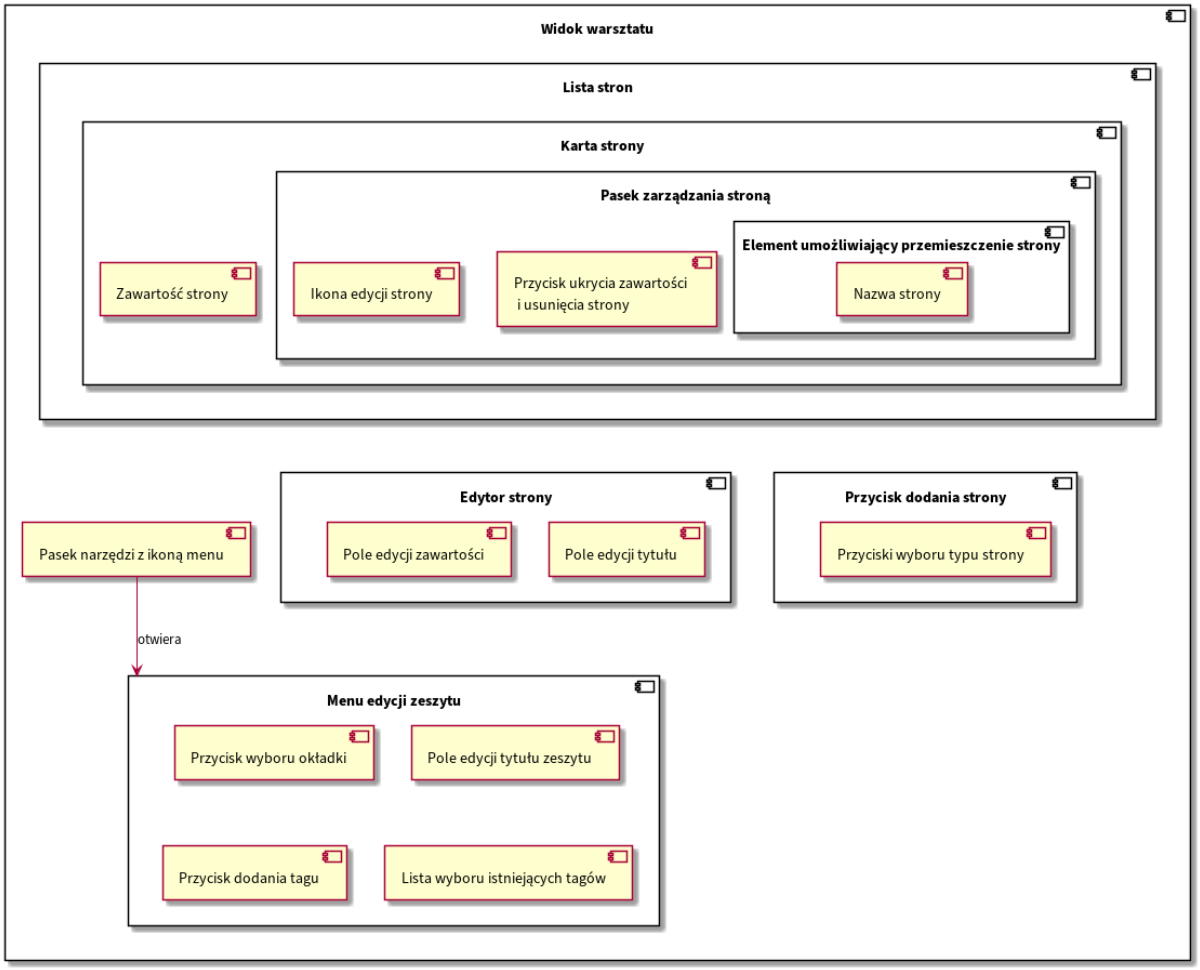
Rys. 12. Widok biblioteki.

Grupa komponentów wspólnych stanowi elementy, z których korzystają oba widoki — biblioteka i warsztat. Należą do nich: AddAlert — umożliwiający dodanie elementu poprzez wpisanie jego nazwy, FloatingButton — obecny w postaci przycisku dodania elementu, a także komponenty odpowiadające za przypisanie określonego stylu komponentom w nich zagnieżdżonych.

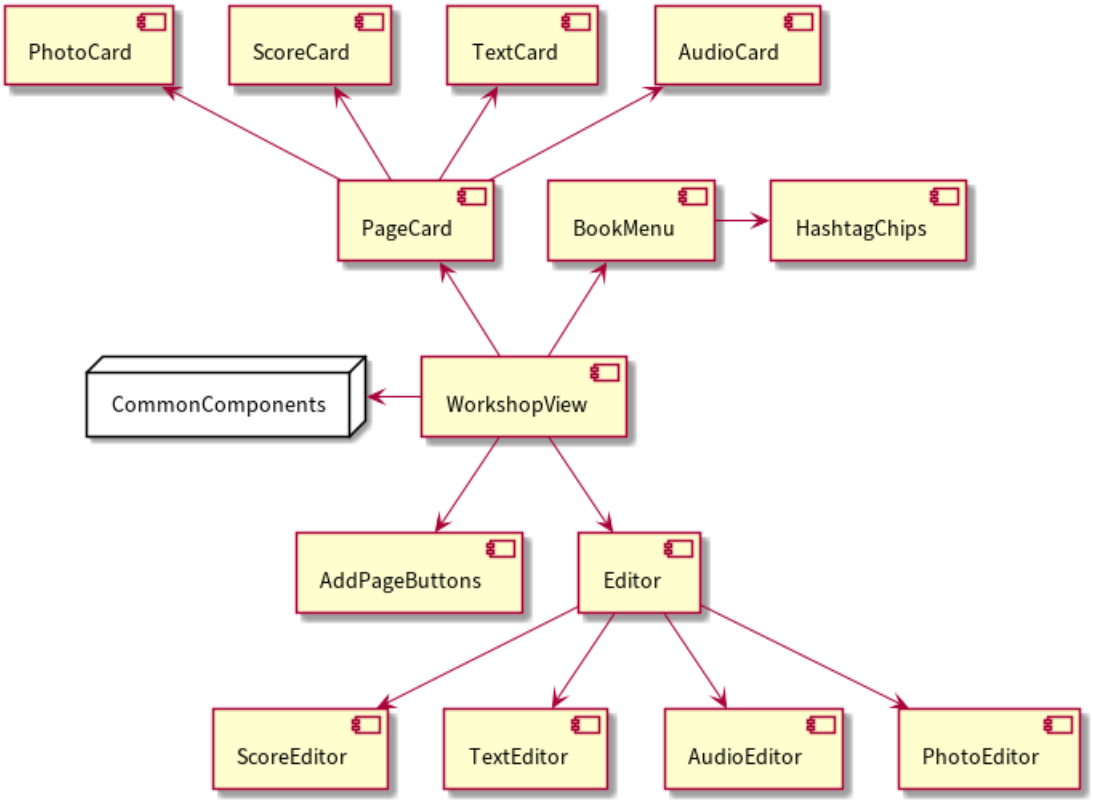
Widok warsztatu umożliwia wizualizację i edycję zawartości zeszytu. Przycisk w rogu ekranu pozwala dodać stronę do zeszytu. Po dodaniu jest ona reprezentowana przez komponent karty. Zawartość strony może zostać ukryta, przy użyciu przycisku z ikoną oka [15]. Przytrzymanie przycisku powoduje trwałe usunięcie strony, sygnalizowane wibracją urządzenia. Strona zawiera także przycisk umożliwiający jej przemieszczenie względem pozostałych stron. Jest to realizowane przy użyciu biblioteki SortableJS [33]. W tym miejscu wyświetla się także nazwa strony, jeżeli została nadana przez użytkownika.

Możliwość edycji strony realizowana jest przez modal, otwierany po kliknięciu ikony plusa [15]. Zawartość modalu zależy od typu edytowanej strony, zawierając stosowny interfejs edycji zawartości. Każdej stronie można tu również przypisać nazwę.

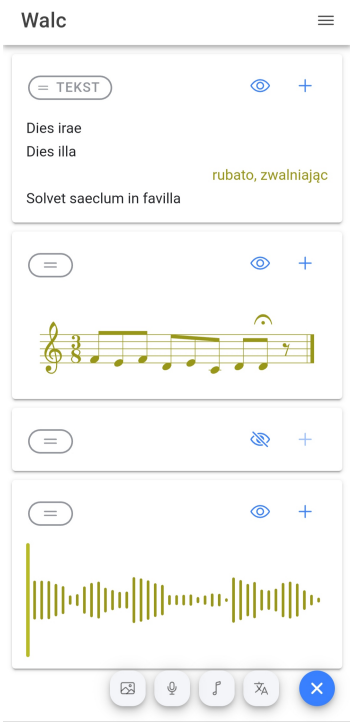
Modyfikacja danych zeszytu umożliwiana jest przez menu, otwierane przy kliknięciu górnego paska widoku zeszytu. Przycisk z ikoną zdjęcia [15] umożliwia dodanie okładki; kliknięcie symbolu zakładki pozwala przypisać słowo kluczowe do zeszytu. Można w tym miejscu również zmienić tytuł zeszytu.



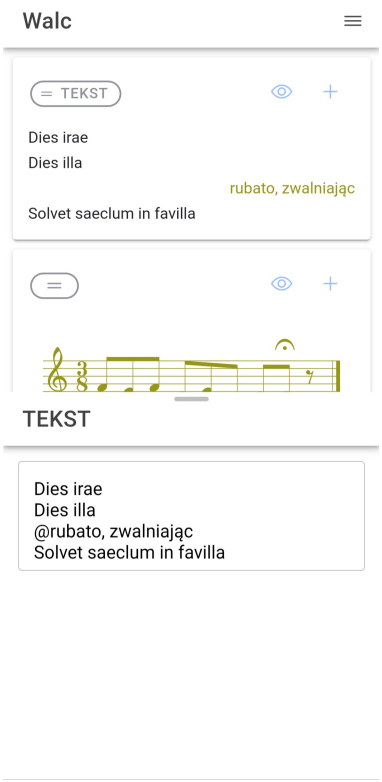
Rys. 13. Diagram interfejsu widoku warsztatu.



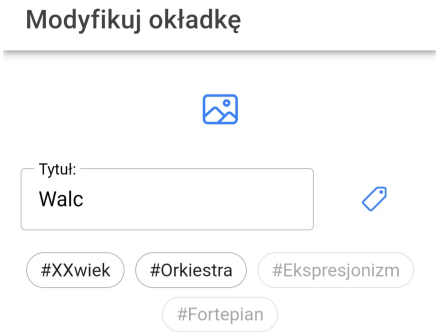
Rys. 14. Drzewo komponentów Vue realizujących widoku zeszytu.



Rys. 15. Widok warsztatu.



Rys. 16. Widok warsztatu z otworzonym edytorem tekstu.



Rys. 17. Widok menu modyfikacji zeszytu.

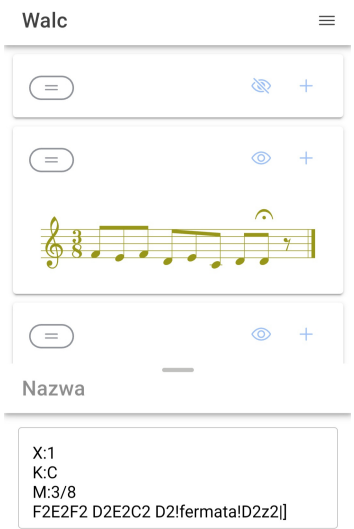
3.6.2. Implementacja funkcjonalności wspierających aranżację

Przedstawione wcześniej komponenty stanowią „środowisko", które umożliwia 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 [38].

Komponenty wizualizacji i edycji tekstu obsługują notatki tekstowe, umożliwiając dodawanie subtelnych (jako: nieodwracających uwagi) komentarzy. Podczas edycji użytkownik zapisuje notatkę w polu tekstowym [15]. Linijka tekstu poprzedzona symbolem „@" traktowana jest jako komentarz. Aby ułatwić tworzenie komentarzy, układ klawiatury przy edycji treści takiej strony ustawiony jest na tryb *e-mail* [15]. Komentarze wyświetlane są przy prawej krawędzi strony, mniej wyrazistą czcionką. Zostało to udokumentowane na rysunku nr 16.

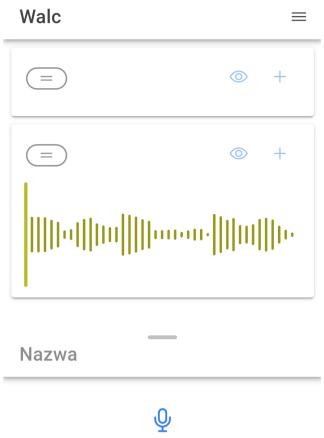
Wizualizacja zapisu nutowego realizowana jest przy użyciu biblioteki abcjs [1]. W ramach komponentu umożliwiającego edycję użytkownik znów zapisuje notatkę w polu tekstowym, lecz jest ona renderowana jako zapis nutowy w komponencie strony. Pomocna jest tu forma modala, gdyż umożliwiaja on śledzenie zmian zapisu nutowego podczas jego edycji. Z uwagi na przystępną zasadę działania biblioteki abcjs — 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 abcjs.



Rys. 18. Edycja zapisu nutowego.

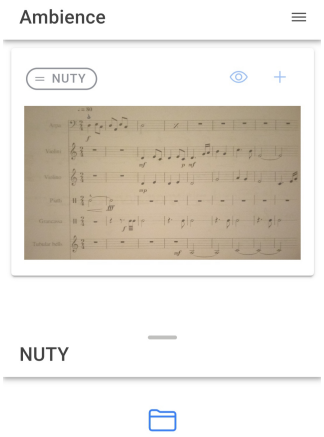
Komponent edycji notatek dźwiękowych wykorzystuje do działania wtyczkę capacitor-

voice-recorder [6]. Pozwala ona nagrać dźwięk, korzystając z mikrofonu urządzenia. Po nagraniu użytkownik może ponownie edytować zawartość strony, zastępując wcześniejszy zapis dźwiękowy nowym nagraniem. Wizualizacja fali dźwiękowej, realizowana przez bibliotekę wavesurfer.js [39]. Umożliwia to orientację w jej przebiegu oraz odtwarzanie nagrania od wybranego momentu.



Rys. 19. Edycja strony zawierającej nagranie.

Możliwość użycia zdjęcia jako notatki realizowana jest przez wtyczkę capacitor-file-picker [5], pozwalającej wybrać plik (tu: zdjęcie) z pamięci urządzenia. Wybrane zdjęcie jest wyświetlane jako zawartość strony. Edycja strony polega na możliwości zmiany wybranego zdjęcia.



Rys. 20. Edycja strony zawierającej zdjęcie.

3.7. Continous Integration i Continous Delivery

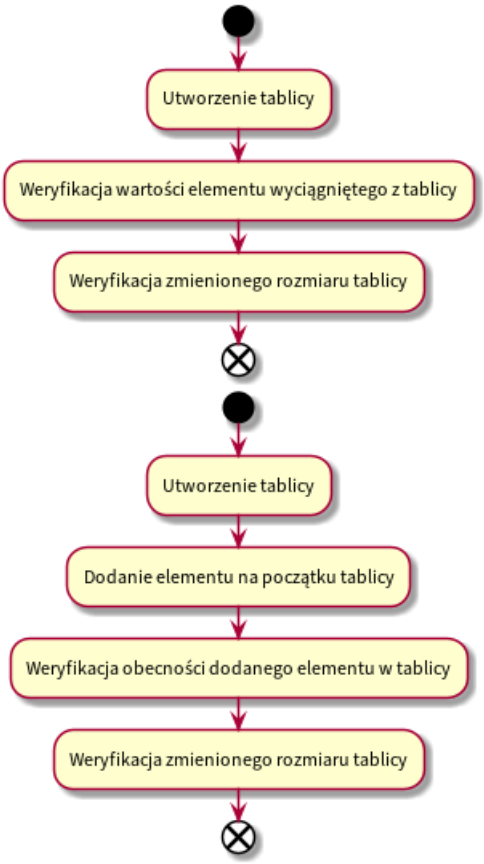
Przeprowadzenie procesu budowy aplikacji i generowania pliku instalacyjnego APK dla platformy Android jest realizowane przez aplikację Buddy CI/CD [3]. Umożliwia to automatyzację tych zadań, przyspieszając proces powstawania aplikacji. W serwise zastosowano pojedynczy pipeline, który wykonuje następujący algorytm:

Rys. 21. Algorytm Continous Delivery realizowany przez Buddy CI/CD.

Po przygotowaniu systemu pipeline projekt jest budowany przy użyciu Ionic [15] oraz Capacitor [4], do wyjściowego projektu platformy mobilnej. Zadeklarowane w osobnym pliku wymagane uprawnienia aplikacji, pozwalające m.in. na korzystanie z mikrofonu urządzenia, są przenoszone do projektu. Wówczas ustawiana jest nazwa wyjściowego pliku instalacyjnego APK. Za jego budowę odpowiada akcja oparta o Gradle [12]. Powstały w ten sposób instalacyjny aplikacji jest przenoszony do lokalizacji root projektu, podpisywany certyfikatem oraz publikowany na repozytorium GitHub [11] przez akcję push.

Została przygotowana także poglądowa implementacja testów jednostkowych, przy uży-

ciu biblioteki Jest [17]. Testowaniu podlega serwis pomocniczy ArrayHelper udostępniający metody modyfikacji tablic. Przebieg algorytmu testów można przedstawić za pomocą diagramu:



Rys. 22. Algorytm testów w ramach Continous Integration realizowany przez Buddy CI/CD.

Algorytmy Countinous Integration i Continous Development zostały zrealizowane w konwencji IAC [19], w pliku YAML [41].

4. Analiza otrzymanych wyników i wnioski

Zrealizowany projekt aplikacji odpowiada na problematykę procesu aranżacji, stanowiąc unikatowe narzędzie wielofunkcyjne, jako organizer koncepcji muzycznych. Możliwości oferowane przez aplikacje wspierają części procesu twórczego pomijane przez dostępne współcześnie oprogramowanie. Będąc najbliżej warsztatu kreatywnego, jako pierwsze „płótno" umożliwiająca odnotowywanie koncepcji, aplikacja dąży do stania się istotnym narzędziem muzyka. W ramach projektu udało się zrealizować założone wymagania:

1. Aplikacja jest mobilna, działając na platformie Android.
2. Przechowuje dane w pełni lokalnie na urządzeniu, oferując pełny zakres funkcjonalności niezależnie od dostępu do internetu.
3. Projekt interfejsu jest zgodny z założeniami, realizowany zgodnie z ideą minimalizmu i użyteczności.
4. Aplikacja umożliwia zapis oraz rewizję pomysłów i koncepcji w formie notatek — textit-stron.
5. Tworzone przez użytkownika strony mogą być realizowane przy użyciu różnych rodzajów notacji oraz jako multimedia.
6. Strony mogą być organizowane wizualnie || przestrzennie — w określonej kolejności, umożliwiając ukrywanie zawartości.
7. Zeszyty, stanowiące zbiory stron, mogą organizowane przy użyciu słów kluczowych oraz przeszukiwane — zarówno po słowach kluczowych, jak i tytułach.
8. Wybrane przez użytkownika zeszyty mogą być opatrzone okładką, ułatwiającą ich identyfikację.

Jednocześnie dostępne funkcjonalności wciąż mogą być rozbudowywane w wielu kierunkach, zwiększając możliwości aplikacji oraz swobodę użytkownika podczas procesu twórczego:

1. Rozszerzenie funkcjonalności istniejących typów stron: umożliwiając odtworzenie zapisu nutowego, modyfikację nagrania dźwiękowego.

2. Możliwość dodawania innych typów stron — na przykład załączników PDF.
3. Umożliwienie eksportu i importu tworzonych zeszytów oraz mechanizmy kopii zapasowych.
4. Architektura umożliwiająca połączenie z internetem celem synchronizacji danych pomiędzy urządzeniami.

Pozostają jednak wciąż pewne kwestie wymagające weryfikacji. Chociaż nie zauważono istotnych błędów działania podczas użytkowania aplikacji, okres jej testowania jest zbyt krótki, aby móc poświadczyć jej niezawodność. Problematyczny może się okazać mechanizm persystencji, mogąc powodować nieprzewidziane zdarzenia przy wyłączeniu aplikacji przed zakończeniem przebiegu funkcji realizujących zapis danych w bazie. W takiej sytuacji można temu wciąż zaradzić, rozbudowując architekturę serwisów persystencji lub zmieniając metodę zapisu danych. Praca aplikacji może również przebiegać zawodnie na urządzeniach starszych, z uwagi na spadek wydajności. Skomplikowana problematyka optymalizacji wykracza jednak poza zakres tej pracy.

Kod źródłowy aplikacji umożliwiający jej budowę oraz zbudowany instalacyjnym APK zostały umieszczone na płycie dołączonej do pracy. Są one również dostępne na publicznym repozytorium GitHub na gałęzi enquotepraca-dyplomowa [20].

Bibliografia

[1] *abcjs*, URL: https://github.com/paulrosen/abcjs, dostęp: 01.01.2024.

[2] *Base64*, URL: https://developer.mozilla.org/en-US/docs/Glossary/Base64, dostęp: 01.01.2024.

[3] *Buddy*, URL: https://buddy.works/docs, dostęp: 01.01.2024.

[4] *capacitor*, URL: https://capacitorjs.com/docs, dostęp: 01.01.2024.

[5] *Capacitor Plugins*, URL: https://github.com/capawesome-team/capacitor-plugins, dostęp: 01.01.2024.

[6] *Capacitor Voice Recorder*, URL: https://github.com/tchvu3/capacitor-voice-recorder, dostęp: 01.01.2024.

[7] Jerse T.A. Dodge C., *Computer Music: Synthesis, Composition, and Performance*, Schirmer Books, 1997.

[8] *Evernote*, URL: https://evernote.com/, dostęp: 01.01.2024.

[9] Skowron Z. Fubini E., *Historia estetyki muzycznej*, Studia et Dissertationes Instituti Musicologiae Universitatis Varsoviensis, Musica Iagellonica, 1997.

[10] *git*, URL: https://git-scm.com/doc, dostęp: 01.01.2024.

[11] *GitHub*, URL: https://github.com/, dostęp: 01.01.2024.

[12] *Gradle*, URL: https://gradle.org/, dostęp: 01.01.2024.

[13] Burton S.L. Greher G.R., *Creative Music Making at Your Fingertips: A Mobile Technology Guide for Music Educators*, Oxford University Press, 2021.

[14] Seago A. Holland S. Wilkie K., *Music and Human-Computer Interaction*, Springer London, 2015.

[15] *Ionic Framework*, URL: https://ionicframework.com/docs, dostęp: 01.01.2024.

[16] *Ionic Storage*, URL: https://github.com/ionic-team/ionic-storage, dostęp: 01.01.2024.

[17] *Jest*, URL: https://jestjs.io/, dostęp: 01.01.2024.

[18] *JSON*, URL: https://json.org/, dostęp: 01.01.2024.

[19] Morris K., *Infrastructure as code*, O'Reilly Media, 2020.

[20] *Kod źródłowy*, URL: https://github.com/mjakobsche/ConceptSound/tree/praca-dyplomowa, dostęp: 01.01.2024.

[21] *Logseq*, URL: https://docs.logseq.com/, dostęp: 01.01.2024.

[22] Kowalska M., *ABC historii muzyki*, Musica Iagiellonica, 2005.

[23] Collins N., *Introduction to Computer Music*, Wiley, 2010.

[24] *Node.js*, URL: https://nodejs.org/docs/latest/api, dostęp: 01.01.2024.

[25] *Notion*, URL: https://www.notion.so/, dostęp: 01.01.2024.

[26] *npm*, URL: https://docs.npmjs.com/, dostęp: 01.01.2024.

[27] *Obsidian*, URL: https://obsidian.md/, dostęp: 01.01.2024.

[28] *OneNote*, URL: https://www.onenote.com/, dostęp: 01.01.2024.

[29] Manning P., *Electronic and Computer Music*, OUP USA, 2013.

[30] *Pinia*, URL: https://pinia.vuejs.org/, dostęp: 01.01.2024.

[31] Żmigrodzki P. red., *Wielki słownik języka polskiego PAN*, 2007, URL: https://wsjp.pl, dostęp: 01.01.2024.

[32] Adler S., *The Study of Orchestration*, The Study of Orchestration, W.W. Norton, 2016.

[33] *Sortable*, URL: https://github.com/SortableJS/Sortable, dostęp: 01.01.2024.

[34] Rutherford-Johnson T., *Music after the fall: Modern composition and culture since 1989*, University of California Press, 2017.

[35] *TypeScript*, URL: https://www.typescriptlang.org/docs/, dostęp: 01.01.2024.

[36] Michels U., *Atlas Muzyki*, t. 1, Prószyński Media, 2002.

[37] Michels U., *Atlas Muzyki*, t. 2, Prószyński Media, 2003.

[38] *Vue.js*, URL: https://vuejs.org/, dostęp: 01.01.2024.

[39] *wavesurfer.js*, URL: https://wavesurfer.xyz/docs/, dostęp: 01.01.2024.

[40] *WebStorm*, URL: https://www.jetbrains.com/webstorm, dostęp: 01.01.2024.

[41] *YAML*, URL: https://yaml.org/, dostęp: 01.01.2024.

Spis rysunków

1. Modele encji implementujące interfejs Entity. 19
2. Model obiektu Index. 20
3. Diagram przepływu dla widoku biblioteki zeszytów użytkownika. 21
4. Diagram przepływu dla menu edycji zeszytu. 21
5. Diagram przepływu dla widoku warsztatu użytkownika. 22
6. Diagram realizacji architektury warstwowej. 23
7. Diagram przedstawiający komunikację między serwisami warstw aplikacji. . . 24
8. Serwisy warstwy dostępu do danych. 25
9. Serwisy warstwy logiki biznesowej. 26
10. Diagram interfejsu widoku biblioteki. 28
11. Drzewo komponentów Vue realizujących widok biblioteki. 28
12. Widok biblioteki. 29
13. Diagram interfejsu widoku warsztatu. 30
14. Drzewo komponentów Vue realizujących widoku zeszytu. 31
15. Widok warsztatu. 31
16. Widok warsztatu z otworzonym edytorem tekstu. 32
17. Widok menu modyfikacji zeszytu. 32
18. Edycja zapisu nutowego. 33
19. Edycja strony zawierającej nagranie. 34
20. Edycja strony zawierającej zdjęcie. 34
21. Algorytm Continous Delivery realizowany przez Buddy CI/CD. 35
22. Algorytm testów w ramach Continous Integration realizowany przez Buddy CI/CD. 36