AKADEMIA MARYNARKI WOJENNEJ
im. BOHATERÓW WESTERPLATTE
Wydział Mechaniczno-Elektryczny
Instytut Elektrotechniki i Automatyki okrętowej
Katedra Informatyki

PRACA DYPLOMOWA

INŻYNIERSKA

Temat:

Projekt i implementacja aplikacji mobilnej przeznaczonej do nauki historii sztuki

Wykonawca: Paulina NIEWIAROWSKA

Kierownik pracy	Kierownik Katedry
dr inż. Artur Zacniewski	kmdr dr hab. inż. Andrzej ŻAK, prof. nadzw. AMW

Ocena pracy dyplomowej ……..…… ……...…….

słownie — data i podpis Przewodniczącego Komisji Egzaminacyjnej

Gdynia, dnia …….…….. r.

Paulina Niewiarowska

Imię i Nazwisko

22322

Nr albumu

O Ś W I A D C Z E N I E

Oświadczam, że przedłożoną do egzaminu pracę dyplomową pt. ***Projekt i implementacja aplikacji mobilnej przeznaczonej do nauki historii sztuki*** kończącą studia I stopnia napisałam samodzielnie. Przy wykonywaniu pracy nie zlecałam jej opracowania ani żadnej jej części innym osobom, jak też nie skopiowałam cudzych opracowań i przestrzegałam postanowień Ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (Dz. U. z 2018 r. poz. 1191 z późn. zmianami).

Ponadto oświadczam, iż treści zaczerpnięte z literatury przedmiotu są oznaczone w tekście oraz w przypisach, w sposób ogólnie przyjęty dla prac naukowych.

Jednocześnie przyjmuję do wiadomości, że gdyby powyższe oświadczenie okazało się nieprawdziwe, jestem świadoma zasadności cofnięcia decyzji o wydaniu mi dyplomu.

Wyrażam zgodę na udostępnienie mojej pracy dyplomowej czytelnikom.

Przekazuję moją pracę dyplomową do ogólnego Repozytorium Prac Dyplomowych.

……

podpis osoby składającej oświadczenie

SPIS TREŚCI

WYKAZ WAŻNIEJSZYCH SKRÓTÓW I OZNACZEŃ

Android	„Android jest najpopularniejszą na świecie mobilną platformą systemową (...) o otwartym kodzie źródłowym, bazującą na Linuksie i wspieraną przez Google'a" [1].
IDE	(ang. *Integrated Development Environment*) – zintegrowane środowisko programistyczne. IDE to oprogramowanie służące do tworzenia, modyfikowania, testowania i utrzymywania aplikacji (…). Współczesne IDE są rozbudowanymi narzędziami, składającymi się z zestawu aplikacji (…). Udostępniają złożone funkcjonalności obejmujące edycję kodu źródłowego, testowanie, kompilacje i uruchomienie programu oraz wiele innych możliwości [2].
Android Studio	„Android Studio jest środowiskiem programistycznym służącym do pisania aplikacji (…), która zawiera Android SDK i dodatkowe narzędzia graficzne wspomagające tworzenie aplikacji na Androida" [1].
UML	(ang. *Unified Modelling Language*). „Język UML jest graficznym językiem wizualizacji, specyfikowania, tworzenia i dokumentowania składników systemów informatycznych" [3].
Lambda	Wyrażenia lambda są krótkimi blokami kodu, które mogą przyjmować parametry i zwraca wartość. Wyrażenia lambda przypominają metody, jednak w odróżnieniu od nich nie muszą posiadać nazwy i można je implementować bezpośrednio w ciele większej metody [4].
Widok	(ang. *View*). „Warstwa widoku odpowiedzialna jest za prezentacje danych, stanu systemu i bieżących operacji w interfejsie graficznym, a także za inicjalizację i wiązanie Modelu Widoku (*ViewModel*) z elementami widoku" [5].
Model Widoku	(ang. *ViewModel*). „Warstwa widoku modelu zajmuje się dostarczaniem danych modelu dla warstwy widoku

	oraz podejmowaniem akcji na rzecz wywołanego zdarzenia z widoku” [5].
Model	(ang. *Model*). „Warstwa modelu odpowiada za logikę biznesową, czyli przetwarzanie, przechowywanie, modyfikacje oraz dostarczanie oczekiwanych danych do widoku modelu” [5].
Obserwator	(ang. *Listener, Observer*). „Wzorzec projektowy należący do grupy wzorców czynnościowych. Używany jest do powiadamiania zainteresowane obiekty o zmianie stanu obserwowanego przez nie obiektu” [6].
LiveData	Obserwowalny dysponent danych. „Służy do obserwacji zmian w danych i aktualizowania widoku” [7]. Jest wyposażony w struktury pozwalające mu dostosowywać swoje działanie do cyklu życia aplikacji. [8]
Cykl życia aplikacji	(ang. *Lifecycle*). Cykl życia aplikacji to zbiór „wszystkich możliwych stanów aplikacji, wraz z informacją jak i kiedy następuje zmiana poszczególnych stanów” [9].
Wyciek pamięci	(ang. *Memory leak*). „Szczególny rodzaj niezamierzonego użycia pamięci przez program komputerowy, gdy nie zwalnia on zaalokowanej wcześniej pamięci, która nie jest już mu potrzebna, a może nawet rezerwować nową. Wycieki pamięci są efektem bardzo niepożądanym. Program bowiem zajmuje coraz więcej pamięci, ale nie jest w stanie jej wykorzystać ani zwolnić” [10].
DAO	(ang. *Data Access Object*). „Komponent dostarczający jednolity interfejs do komunikacji między aplikacją a źródłem danych (np. bazą danych czy plikiem). Jest często łączony z innymi wzorcami projektowymi. Dzięki DAO, aplikacja nie musi znać sposobu oraz ostatecznego miejsca składowania swoich danych, a ewentualne modyfikacje któregoś z czynników nie pociągają za sobą konieczności modyfikowania jej kodu źródłowego” [11].

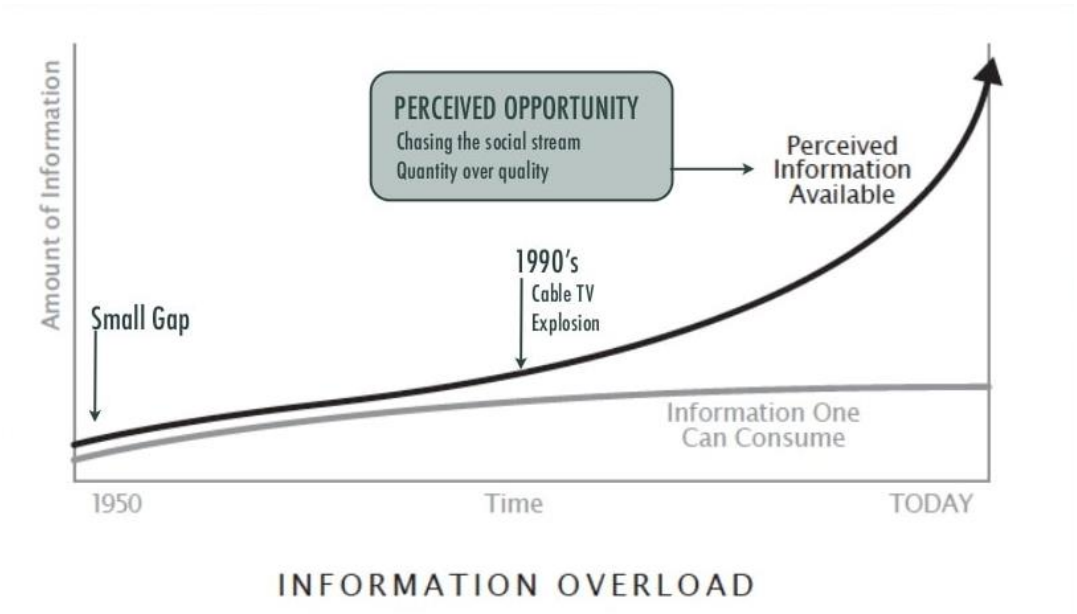
Hermetyzacja	(ang. *encapsulation*). „Jedno z założeń programowania obiektowego. Hermetyzacja polega na ukrywaniu pewnych danych składowych lub metod obiektów danej klasy tak, aby były one dostępne tylko metodom wewnętrznym danej klasy lub funkcjom zaprzyjaźnionym” [12].
ZIP	„ZIP to będący w powszechnym użyciu format pliku archiwum, który jest stosowany do kompresji jednego lub więcej plików w jednym folderze, co pozwala na zmniejszenie ich ogólnego rozmiaru i ułatwia ich przesyłanie” [13].
Intencja	(ang. *Intent*). „Mechanizm ten odpowiedzialny jest przede wszystkim za obsługę rozkazów wydawanych przez Użytkownika. Za pomocą intencji możemy wprowadzić komunikację pomiędzy aplikacjami (lub mniejszymi komponentami, jak Usługi, Aktywności itp.). Jednak najważniejszym zadaniem tego komponentu jest uruchamianie odpowiednich aplikacji/Aktywności” [14].
Bundle	Prosty obiekt służący do przekazywania danych pomiędzy aktywnościami w aplikacji.
Odbiornik emisji	(ang. *Broadcast Receiver*). Komponent pozwalający na odbieranie powiadomień (w postaci intencji) wysyłanych przez inne aplikacje lub serwisy.
Monetyzacja	W przypadku aplikacji mobilnych oznacza wykorzystanie aplikacji w celu zarabiania pieniędzy, co oznacza uczynienie jej płatną, umieszczenie reklam wewnątrz aplikacji, i/lub udostępnienie możliwości zakupów w aplikacji [15].

WSTĘP

"In your thirst for knowledge, be sure not to drown in all the information."
„W swym pragnieniu wiedzy uważaj, by nie utonąć wśród zalewu fal informacji."
— Anthony J. D'Angelo, author, The College Blue Book

Rozwój Internetu oraz technologii doprowadził do sytuacji, w której wiedza na niemal dowolny temat znajduje się (dosłownie) na wyciągnięcie ręki. Jednak otacza nas nie tylko wiedza pożądana i potrzebna. Niestety, coraz częściej informacje ważne i przydatne gubią się w natłoku informacji bezużytecznych i szkodliwych, a nadmiar bodźców i sygnałów (rys. 1), które muszą przetworzyć nasze zmysły oraz mózgi coraz częściej działa na naszą niekorzyść.

Zarysowana tendencja ma swoje odbicie również na rynku aplikacji mobilnych. Sklepy z aplikacjami pęcznieją od programów coraz bardziej wielofunkcyjnych, ale też, co ważniejsze, przeładowanych reklamami i innymi formami marketingu. Coraz częściej spotkać można aplikacje, które udostępniają jedynie część funkcjonalności, te najważniejsze czyniąc płatnymi. Co szczególnie przykre, dotyczy to również sektora aplikacji edukacyjnych oraz tych naukę wspomagających.



Rys. 1 Wykres orientacyjnie prezentujący wzrost ilości informacji, z jaką przeciętny człowiek musi mierzyć się co roku (na przestrzeni ostatnich dziesięcioleci) [16]

Celem niniejszej pracy jest zaprezentowanie projektu oraz implementacji aplikacji mobilnej służącej do nauki historii sztuki. Jest ona pierwszą aplikacją z projektu Mentala, którego celem będzie udostępnienie możliwie szerokiego zestawu aplikacji wspomagających swoich Użytkowników w rozwoju bez powodowania efektu przeciążenia informacjami. Prezentowana aplikacja nosi nazwę HaSZ (jest to specyficzny akronim pochodzący od słów Historia SZtuki, popularnie używany w liceach plastycznych, w których wspomniana dziedzina należy do programu nauczania) i jest przeznaczona na platformę Android. Prezentowana aplikacja dedykowana jest uczniom szkół plastycznych oraz innych kierunków artystycznych, lecz z powodzeniem może zostać wykorzystana również przez nieprofesjonalnych pasjonatów sztuki.

Pierwszy rozdział poświęcony zostanie opisaniu problemu przeciążenia informacyjnego i jego skutków. Zaprezentowana w nim zostanie również charakterystyka prezentowanej aplikacji, cel pracy oraz zestawienie najpopularniejszych rozwiązań o podobnej tematyce, czyli programów mobilnych służących do nauki historii sztuki.

Kolejny, drugi rozdział skupi się na projekcie aplikacji HaSZ. Zawierać będzie wymagania funkcjonalne i niefunkcjonalne dotyczące aplikacji, wybór technologii użytych podczas jej tworzenia oraz opis zastosowanej architektury.

Następnie, w rozdziale trzecim, przedstawiona zostanie implementacja aplikacji z podziałem na moduły i opisem wszystkich ważnych funkcjonalności zawartych w projekcie.

Natomiast podsumowanie zawierać będzie plany rozwojowe aplikacji, wnioski z procesu jej powstawania i krótkie streszczenie całości pracy.

1. PREZENTACJA TEMATU

1.1. Przeciążenie informacyjne i jego skutki

Era, w której żyjemy, nie bez powodu nazwana została erą informacji. Ilość wiedzy, jaką otrzymujemy i przetwarzamy każdego dnia wciąż rośnie, na przestrzeni ostatnich dekad przebijając własne wielokrotności. Jednak okazuje się, że sygnały, które musimy na co dzień przetworzyć, nie pozostają obojętne dla naszego zdrowia - zarówno psychicznego, jak i fizycznego. Choć pojęcie przeciążenia informacyjnego znane jest już od lat 70tych XX wieku, zjawisko, które opisuje, długo było lekceważone [17]. Dopiero ostatnie dwie dekady pokazały, że nie jest to jedynie mglisty wymysł, a faktyczny problem dotykający coraz większej ilości ludzi [18]. Badania neuropsychiatryczne pokazują, że ludzie często korzystający z internetu, a zatem przeładowani informacjami, wykazują wzmożoną podatność na stres, co objawia się:

- uczuciem przemęczenia, problemami ze snem, bólami głowy,

- kłopotami z koncentracją, zmniejszoną efektywnością pracy,

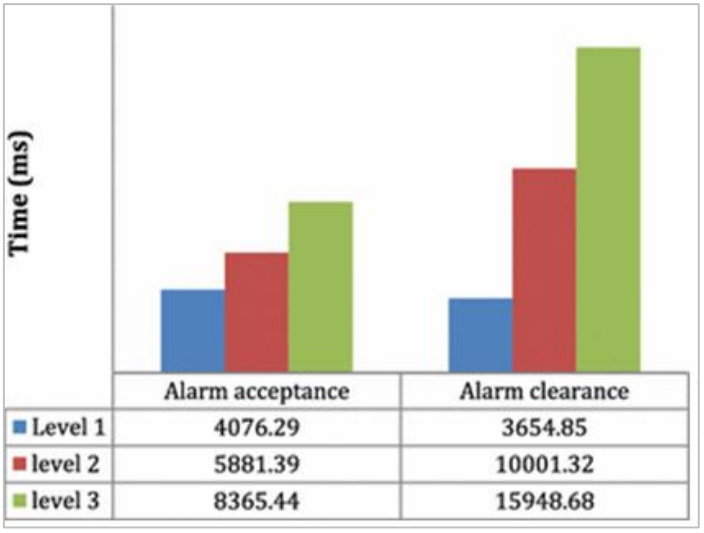
- problemami natury psychologicznej i neurologicznej.

Ostatnia kategoria obejmuje między innymi: podirytowanie, niepokój, zachowania obsesyjno-kompulsywne, problemy z podjęciem decyzji, impulsywność, problemy z pamięcią, i tym podobne [19].

Ponadto zaobserwowano zmiany w strukturach mózgu w stosunku do osób mniej narażonych na nadmiar informacji i bodźców. Stwierdzono także zmiany w sposobie żywienia (objawiające się preferowaniem pożywienia i napojów wysokoenergetycznych i łatwo przyswajalnych) oraz zwiększoną tendencję do oddawania się nałogom [19]. Związane jest to z reakcjami organizmu na przewlekły stres – w takiej sytuacji wymaga on dużej podaży łatwo przyswajalnej energii (układ współczulny aktywowany hormonami stresu między innymi powoduje spowolnienie metabolizmu na rzecz przekierowania energii do ważniejszych organów wewnętrznych i mięśni), natomiast substancje uzależniające zwykle wykazują zdolność uśmierzania napięcia psychicznego.

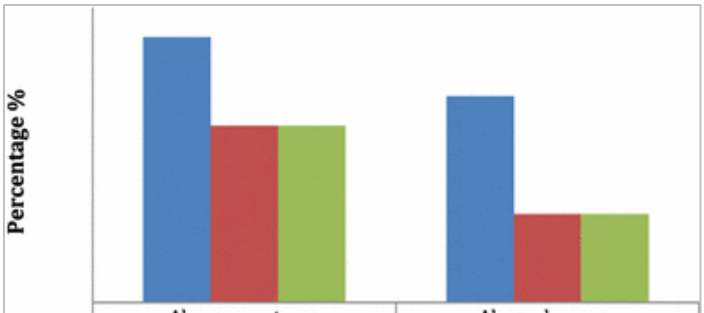
Laboratoryjne badania wykazały, że przeładowanie informacjami powodowało znaczące pogorszenie funkcji kognitywnych a w rezultacie tendencję do podejmowania błędnych decyzji, co jest o tyle niepokojące, że jedno z badań przeprowadzone zostało na przykładzie obsługi alarmów kolejowych w dyspozytorni kolejowej stworzonej na potrzeby

eksperymentu [20]. Z przytoczonego badania bezpośrednio wynika, że nadmiar informacji prowadzi do dłuższego czasu podejmowania decyzji (rys. 2) oraz zwiększa odsetek błędnych decyzji (rys. 3). W zawodach takich jak medycyna, dyspozytornie ruchu kolejowego czy też lotniczego oraz innych profesjach powiązanych bezpośrednio ze zdrowiem i życiem, przeładowanie informacjami stanowi naprawdę poważny problem, który może prowadzić do tragicznych konsekwencji.



	Alarm acceptance	Alarm clearance
Level 1	4076.29	3654.85
level 2	5881.39	10001.32
level 3	8365.44	15948.68

Rys. 2. Porównanie czasów reakcji na alarm w zaimprowizowanej dyspozytorni kolejowej [20]



	Alarm acceptance	Alarm clearance
Level 1	0.99	0.97
Level2	0.96	0.93
Level 3	0.96	0.93

Rys. 3. Procentowa ilość prawidłowych reakcji na alarm [20]

Na obu powyższych rysunkach:

- poziom 1 oznacza obecność jedynie informacji niezbędnych do wykonania zadania,

- kolejne dwa poziomy oznaczają odpowiednio wzrastającą ilość informacji zbędnych i dodatkowych, przy czym poziom 3 oznacza największą ich ilość.

1.2. Cel pracy

Dostęp do informacji jest oczywiście ważny i potrzebny. To nadmiar zbędnych informacji negatywnie wpływa na nasz mózg, a w konsekwencji na całe ciało i życie. Nadmiar informacji oznacza nie tylko ilość konkretnych komunikatów, jakie przetwarzamy w celu zrozumienia ich treści, ale również nadmiar bodźców docierających do naszych zmysłów. Oznacza to, że powodem przeciążenia informacyjnego są nie tylko powszechnie już oczerniane media społecznościowe, ale również zbyt duża ilość światła oraz różnych i wyrazistych kolorów i dźwięków docierających do naszych zmysłów z wielu innych źródeł. W przypadku programów i aplikacji użytkowych oznacza to:

- zbyt skomplikowane interfejsy o zbyt krzykliwej szacie kolorystycznej,
- nadmiar funkcji, wśród których Użytkownik musi szukać jednej lub dwóch, z których faktycznie korzysta,
- reklamy i inne materiały promocyjne (z założenia tworzone w sposób mający przyciągać uwagę, a więc zawierające wszystkie elementy powodujące przeciążenie informacyjne).

Prezentowana aplikacja z pewnością nie rozwiąże globalnego problemu przeciążenia informacjami, wykorzystuje jednak zaprezentowaną powyżej wiedzę w celu udzielenia wsparcia swoim Użytkownikom. Aplikacja HaSZ służy do nauki historii sztuki i temu zadaniu jest bezpośrednio dedykowana. Została w związku z tym zaopatrzona w zestaw cech wspomagających uczącego się Użytkownika, pozbawiono ją natomiast aspektów wpływających negatywnie na i tak już przemęczone umysły Odbiorców. Oznacza to:

- prosty interfejs z ograniczoną i wyciszoną szatą kolorystyczną,
- obecność jedynie funkcji niezbędnych do nauki, brak nadmiarowych i niepotrzebnych funkcjonalności,
- funkcjonalność uczenia się opartą na sposobie nauki typu „fiszki”, w którym Użytkownik sam definiuje materiał, na którym zamierza pracować i posiada pod tym względem pełną dowolność wyboru oraz nadzoru treści,
- absolutny brak reklam, banerów i innych krzykliwych treści.

Dodatkowo, aplikacja jest i zawsze będzie w pełni darmowa, ponieważ według Autorki wiedza, bezpieczny rozwój i edukacja są zbyt ważne, by je monetyzować. Ponadto, pełny kod aplikacji zostanie opublikowany w ramach licencji otwartego oprogramowania.

1.3. Porównanie z podobnymi rozwiązaniami

Na rynku aplikacji istnieje wiele rozwiązań służących de facto temu samemu celowi – czyli nauce historii sztuki. Są to jednak rozwiązania nieprzyjazne Użytkownikom, płatne, przeładowane niepotrzebnymi modułami. Za przykład mogą tu posłużyć najbardziej popularne aplikacje dostępne w sklepie z aplikacjami (Sklep Play):

- **Aplikacja Learn Art History, Artworks & Paintings – Artly**

Tabela 1. Zestawienie cech aplikacji Artly [21]

Aplikacja Artly		
Zalety	Wady	Zrzut ekranu z aplikacji
+ Schludny i względnie prosty interfejs.	- Test jedynie z nazwisk autorów dzieł, brak testu z tytułów, datowania.	
+ Moduł testowy.	- Zbędne powiadomienia.	
+ Podział na epoki.	- Brak możliwości dodania własnego obrazka.	
	- Brak możliwości zdefiniowania własnych epok i nurtów w sztuce.	
	- Konto premium wymagane do nauki większości epok.	
	- Ograniczona ilość prób przy podejmowaniu się testu z wybranej epoki.	
	- Reklamy, w dodatku zajmujące cały ekran i pozbawione możliwości natychmiastowego wyłączenia (wymagają odczekania kilku sekund).	

- **Aplikacja History of Art**

Tabela 2. Zestawienie cech aplikacji History of Art [22]

<table>
<tr><th colspan="3">Aplikacja History of Art</th></tr>
<tr><th>Zalety</th><th>Wady</th><th>Zrzut ekranu z aplikacji</th></tr>
<tr><td>+ Podział na epoki.</td><td>- Krzykliwy i zawierający zbyt dużo elementów interfejs.</td><td rowspan="6"> 
</td></tr>
<tr><td>+ Brak reklam.</td><td>- Nadmiar zbędnych funkcjonalności, wśród których trzeba szukać tej jednej potrzebnej.</td></tr>
<tr><td></td><td>- Brak możliwości dodania własnego obrazka.</td></tr>
<tr><td rowspan="2">+/- nie posiada modułu testowego, w zamian proponuje możliwość układania puzzli z dzieła</td><td>- Brak możliwości zdefiniowania własnych epok i nurtów w sztuce.</td></tr>
<tr><td>- Moduł do nauki niepełny, za pełną wersję trzeba płacić.</td></tr>
<tr><td></td><td>- Większość treści niedostępna bez wykup ienia.</td></tr>
<tr><td></td><td>- brak reklam zastąpiony dużymi i krzykliwymi propozycjami wykupienia konta premium.</td></tr>
</table>

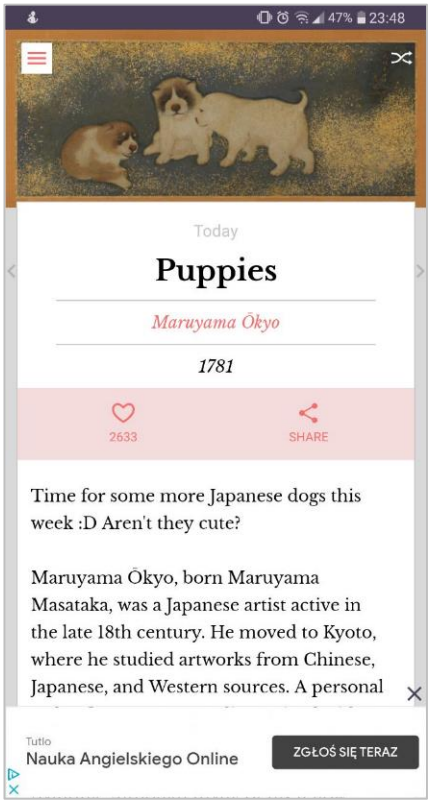
- **Aplikacja Google Arts & Culture**

Tabela 3. Zestawienie cech aplikacji Google Arts & Culture [23]

Aplikacja Google Arts & Culture		
Zalety	Wady	Zrzut ekranu z aplikacji
+ Prawdopodobnie może posłużyć jako baza ciekawostek.	- Ciężko cokolwiek o aplikacji stwierdzić, gdyż jest zdecydowanie przeładowana rozwiązaniami. Główny ekran aplikacji można dowolnie długo przesuwać w dół, a wciąż pojawiają się nowe kolorowe funkcje i informacje.	
	- Raczej nie może służyć do nauki historii sztuki, choć ciężko cokolwiek stwierdzić z całą pewnością, gdyż funkcjonalności przesłaniają się wzajemnie i odnalezienie tej właściwej graniczy z cudem.	
	- Raczej brak możliwości dodawania własnych obrazków i definiowania własnych elementów periodyzacji – to, czy taki moduł istnieje, również ciężko stwierdzić, przez ich nadmiar i brak podziału.	

- **Aplikacja DailyArt**

Tabela 4. Zestawienie cech aplikacji DailyArt [24]

Aplikacja DailyArt		
Zalety	Wady	Zrzut ekranu z aplikacji
+ pokazuje codziennie inne dzieło sztuki i jego opis.	- Brak podziału na epoki i nurty. Brak możliwości zdefiniowania własnych epok i nurtów w sztuce.	
	- Posiada wiele niepotrzebnych funkcjonalności.	
	- Brak możliwości dodania własnego obrazka.	
	- Nie posiada modułu sprawdzania wiedzy.	
	- Aby dostać się do części funkcji należy wykupić usługę premium.	
	- Aplikacja zasypuje bodźcami i informacjami.	
	- Duża ilość reklam, w tym reklamy pełnoekranowe, na możliwość ich wyłączenia należy czekać co najmniej kilka sekund.	

- **Aplikacja Artier**

Tabela 5. Zestawienie cech aplikacji Artier [25]

Aplikacja Artier		
Zalety	Wady	Zrzut ekranu z aplikacji
+ Podział na wiele kategorii.	- Wyszukiwanie w kategoriach zamiast epok. Należy z bardzo długiej, alfabetycznej listy nurtów wybrać ten jeden, który nas interesuje.	
+ Prosta w obsłudze.	- Brak podziału na epoki.	
+ Brak reklam.	- Brak możliwości dodania własnego obrazka.	
	- Nie posiada modułu sprawdzania wiedzy.	
	- Brak sztuki prehistorycznej i bardzo skąpy zasób antycznej.	
	- Brak możliwości sprawdzenia wiedzy.	

Z powyższych opisów wynika, że żadna z aplikacji nie spełnia podstawowego założenia aplikacji HaSZ - żadna bowiem nie pozwala na dodawanie własnych obrazków oraz definiowanie własnych epok i nurtów w sztuce. Tymczasem studentom i uczniom szkół plastycznych właśnie taka funkcjonalność jest najbardziej potrzebna – dostają oni bowiem zestaw obrazków od nauczycieli prowadzących zajęcia i ich zakres nauki obejmuje dokładnie te materiały.

Ponadto, większość dostępnych aplikacji zawiera reklamy, a treści udostępnia jedynie za opłatą. W sytuacji ekonomiczniej, w jakiej zwykle znajdują się uczniowie i studenci, łatwiej jest im korzystać z dotychczasowych sposobów uczenia się historii sztuki – co oznacza

drukowanie miniatur obrazków i zamieszczanie ich opisów z tyłu wydruku [doświadczenie własne, obserwacja rówieśników podczas nauki].

Z wymienionych aplikacji dwie pozwalają na interakcję z obrazkami:

- History of Art pozwala na układanie puzzli z wybranego dzieła sztuki. Takie podejście w formie zabawy pomaga Użytkownikowi zapoznać się ze szczegółami dzieła.

- Artly sprawdza wiedzę Użytkowników w formie quizu. Choć quiz jest ograniczony do jedynie autora dzieła, sama idea quizu zamiast testu pisanego jest zdecydowanie warta uwagi.

Oba rozwiązania znajdują się na liście funkcjonalności rozwojowych zaplanowanych na kolejne aktualizacje aplikacji HaSZ i w przyszłości zostaną w niej zaimplementowane.

Podsumowując, aplikacja HaSZ jako jedyna z dostępnych posiada możliwość dodawania własnych fiszek z obrazkami, pozwala też na dodawanie i edycję elementów periodyzacyjnych, co zapewnia Użytkownikowi pełną dowolność i kontrolę nad zawartością aplikacji. Dodatkowo, HaSZ posiada bazę paczek gotowych do pobrania, udostępnianych całkowicie za darmo, w związku z czym Użytkownicy nie posiadający własnych zestawów z obrazkami będą mogli zapełnić swoją bazę domyślnym ich pakietem. W przeciwieństwie do większości dostępnych na rynku aplikacji mobilnych w ogóle, HaSZ nie wspiera i nie będzie wspierał reklam ani płatnych usług. W związku z tym aplikacja HaSZ stanowić będzie doskonałe rozwiązanie dla docelowych Użytkowników - najprawdopodobniej lepsze od wszystkich aktualnie dostępnych.

2. PROJEKT APLIKACJI

Projekt aplikacji tworzony jest przed jej implementacją. Służy on zdefiniowaniu i uporządkowaniu wymagań dotyczących aplikacji, tak, aby spełniała ona swoje założenia i wykonywała oczekiwaną od niej pracę. Gdy zebrane zostaną już wymagania funkcjonalne oraz niefunkcjonalne, kolejnym krokiem jest decyzja dotycząca wyboru technologii i bibliotek, które będą potrzebne do realizacji wyznaczonych celów. Poniżej opisane są wszystkie etapy projektowania aplikacji HaSZ.

2.1. Wymagania funkcjonalne

Pierwszym etapem projektowania aplikacji mobilnej HaSZ było zebranie wymagań dotyczących jej funkcjonalności, co oznacza określenie grupy Odbiorców, zakresu działania aplikacji, spodziewanego sposobu jej wykorzystania oraz oczekiwań dotyczących udostępnianych przez aplikację możliwości.

Wymagania funkcjonalne „opisują funkcje (czynności, operacje, usługi) wykonywane przez system" [26]. Wymagania te definiuje się poprzez wyznaczenie efektów działania danych funkcji systemu, z pominięciem wszelkich szczegółów dotyczących realizacji owych funkcji. Język UML posiada typ diagramu dedykowany gromadzeniu i prezentowaniu tego typu wymagań – są to diagramy przypadków użycia [3].

Diagram przypadków użycia ma na celu graficzną reprezentację wszystkich wymagań funkcjonalnych systemu, a prezentuje je w postaci interakcji Aktora (w przypadku aplikacji HaSZ Aktorem jest Użytkownik systemu) z systemem. Każda taka interakcja to oddzielny przypadek użycia zaprezentowany za pomocą owalnych pól z nazwą, związki pomiędzy nimi są natomiast reprezentowane przy pomocy dedykowanych strzałek.

2.1.1. Moduły logiczne

Aplikacja HaSZ składa się z pięciu modułów logicznych. Są to:

- **Moduł Zarządzania Elementami Periodyzacji**

Służy on do administrowania trzema typami elementów: **Epoką** sztuki, **Nurtem** lub okresem w sztuce (który jest zawsze zależny od Epoki, w której się znajduje) oraz **Typem** dzieła (na przykład: malarstwo, rzeźba, ceramika, architektura).

Elementy Periodyzacji muszą znajdować się pod pełną kontrolą Użytkownika aplikacji, co oznacza, że musi on móc je **wyświetlać**, **dodawać**, **edytować** i **usuwać**. Każdemu elementowi periodyzacji przysługuje możliwość zaopatrzenia go w **Ciekawostkę**, których wyświetlanie i edycja również muszą być dostępne z poziomu podglądu elementu periodyzacji lub jego zawartości w Module Uczenia Się.

- **Moduł Zarządzania Obrazkami**

Służący zawiadywaniu Obrazkami znajdującymi się w aplikacji.

Obrazki mogą znaleźć się w aplikacji na dwa sposoby – mogą zostać dodane z pamięci urządzenia końcowego lub zostać pobrane z internetu w formie paczki w Module Pobierania Zasobów.

Każdy Obrazek dodawany do aplikacji musi zostać opatrzony **metadanymi** w postaci:

- określenia **Typu** dzieła i jego przynależności periodyzacyjnej, co oznacza wybór **Epoki i** ewentualnie Nurtu (nie każde dzieło przynależy do konkretnego nurtu, nie może być on zatem wymagany) z kolekcji elementów zdefiniowanych w Module Zarządzania Elementami Periodyzacji. W przypadku braku poszukiwanego elementu, okno wyboru powinno również pozwalać na dodanie nowych elementów periodyzacji.

- ponadto do niezbędnych metadanych każdego obrazka należą: **Nazwa** dzieła, **Autor** dzieła, **Datowanie** dzieła oraz **Lokalizacja** dzieła.

W przypadku własnoręcznego dodawania Obrazków, Użytkownik samodzielnie decyduje, co powinno znaleźć się w każdym z wymienionych pól.

Dodatkowo ma istnieć możliwość wyświetlenia i dodania oraz edycji **Ciekawostek**, po jednej na każde dzieło sztuki.

Nad wszystkimi obrazkami Użytkownik ma pełną kontrolę.

- **Moduł Pobierania Zasobów**

Pełni funkcję menedżera pobierania przygotowanych wcześniej paczek zawierających pakiety gotowych, opisanych przez Autorkę Obrazków.

Użytkownik powinien dostać możliwość **wyboru** dostępnych paczek oraz możliwość **pobrania** ich poprzez kliknięcie wybranej opcji.

Po pobraniu oczekuje się, że Użytkownik będzie mógł w pełni korzystać z zawartości pakietów nie tylko poprzez Moduł Uczenia Się, lecz także przy pomocy Modułu Zarządzania Obrazkami.

- **Moduł Uczenia Się**

W zasadzie pełniący funkcję galerii obrazków uzupełnionej o dodatkowe funkcjonalności i możliwości.

Moduł ma zawierać **okna** pozwalające na wybór Epoki oraz Nurtu w sztuce, których zawartość powinna zostać wyświetlona w formie galerii obrazów.

Galeria obrazów powinna wyświetlać Obrazki przynależące do wybranego elementu periodyzacyjnego w formie **przyciemnionej lub domyślnej** – jasność Obrazka ma zależeć od poziomu jego znajomości prezentowanego przez Użytkownika. Dodatkowo widok galerii powinien udostępniać możliwość podglądu Ciekawostki dotyczącej elementu periodyzacyjnego, którego zawartość jest właśnie prezentowana.

Ponieważ Moduł zakłada naukę, a nie jedynie przeglądanie zawartości aplikacji, w oknie wyświetlania konkretnego obrazka jego metadane powinny być wstępnie **ukryte**, z możliwością **odkrycia ich poprzez kliknięcie**. Po zapoznaniu się z treścią metadanych Użytkownik musi posiadać możliwość poinformowania aplikacji o aktualnym poziomie znajomości obrazka – do tego mają służyć dwa przyciski opisane „**Umiem to**” oraz „**Nie umiem tego**”. Poziom wiedzy można w łatwy sposób oszacować na podstawie widoczności wybranego obrazka z poziomu galerii, nie oczekuje się więc wyświetlania dodatkowych informacji w tym zakresie.

Okno podglądu Obrazka ma również udostępniać możliwość **wyświetlenia Ciekawostki** dotyczącej oglądanego właśnie dzieła oraz możliwość **przejścia do okna edycji** dzieła (Moduł Zarządzania Obrazkami).

- **Moduł Testowy**

Służy do przeprowadzania testu znajomości obrazków.

W celu podjęcia testu należy mieć możliwość **wyboru** Epok / Nurtów, z których test ma zostać przeprowadzony. Wybór powinien dotyczyć **wszystkich** aktualnie dostępnych w aplikacji elementów periodyzacyjnych i pozwalać na ich „**mieszanie**", czyli wybór różnych Nurtów z różnych Epok oraz wybór kilku różnych Epok.

Następnie powinno wyświetlić się okno testu, zawierające kolejno pojedyncze Obrazki z puli wszystkich należących do wybranych elementów periodyzacji, przy czym, co ważne, Obrazki te mają pokazywać się **losowo**.

Każdy taki element wymaga wpisania **wszystkich podstawowych informacji** na jego temat w odpowiednie pola (oznacza to wpisanie: Nazwy, Autora, Datowania, Lokalizacji dzieła oraz ewentualnie Nurtu, do którego przynależy dzieło, o ile do jakiegoś przynależy). Okno z obrazkiem zawiera również **Ciekawostkę** dotyczącą aktualnie pokazywanego dzieła sztuki i pozwala ją wyświetlić, jednak nie wymaga jej znajomości ani uzupełniania danych na jej temat.

Następnie można z**atwierdzić wybór** lub **wyczyścić pola wpisywania** przy pomocy odpowiednich **przycisków** dostępnych na ekranie. W przypadku wyboru opcji wyczyszczenia pól, wszelkie wpisane informacje mają zniknąć, a pola mają oczekiwać na nowy wpis od Użytkownika.

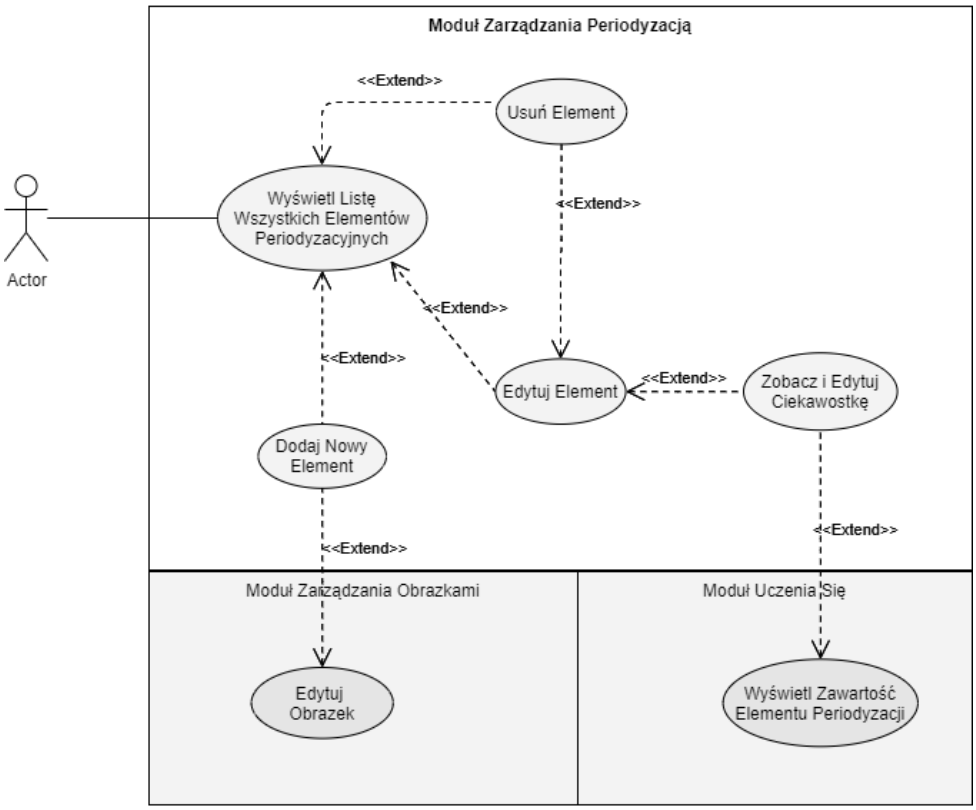
Przycisk zatwierdzenia natomiast ma wysłać odpowiedzi do Modułu, którego zadaniem jest udostępnienie Użytkownikowi i**nformacji zwrotnej** w postaci podświetlenia pól – odpowiednio na czerwono w przypadku nieprawidłowo wpisanej informacji oraz na zielono w przypadku informacji wpisanej prawidłowo. W tym momencie w obu przypadkach musi istnieć możliwość **podglądu** odpowiedzi prawidłowej i samodzielnego porównania jej z wpisanymi danymi – dzięki temu Użytkownik samodzielnie może zdecydować, czy posiada wystarczającą wiedzę na temat wybranego dzieła sztuki, czy też nie.

Kolejne wciśnięcie przycisku zatwierdzenia powinno przenosić Użytkownika do **kolejnego elementu** testu, natomiast wciśnięcie go przy ostatnim elemencie testu powinno skutkować wyświetleniem **podsumowania testu** w postaci wyniku: ilość prawidłowo opisanych obrazków / ilość wszystkich obrazków w teście.

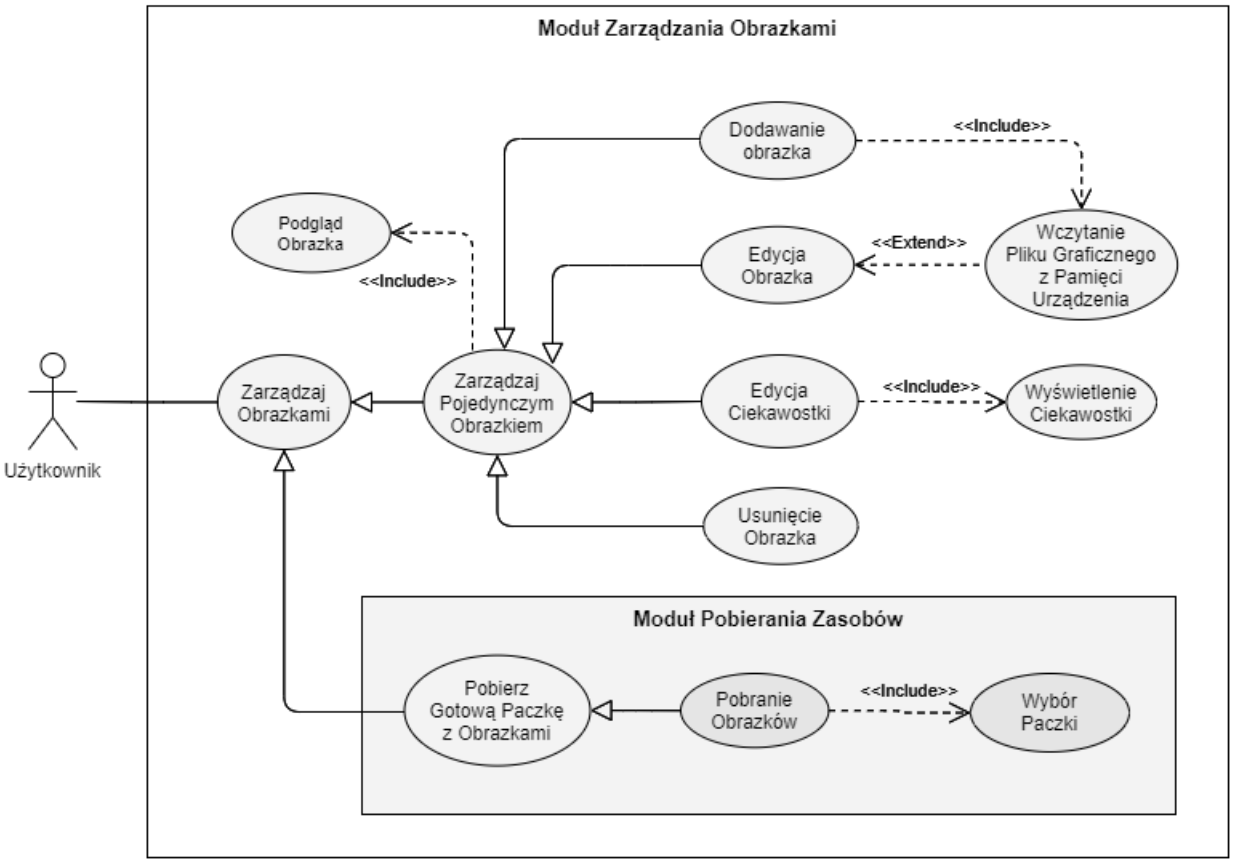
Z okna podsumowania testu Użytkownik musi mieć możliwość **powtórzenia** testu lub **wyjścia** z Modułu Testowego.

2.1.2. Diagramy przypadków użycia

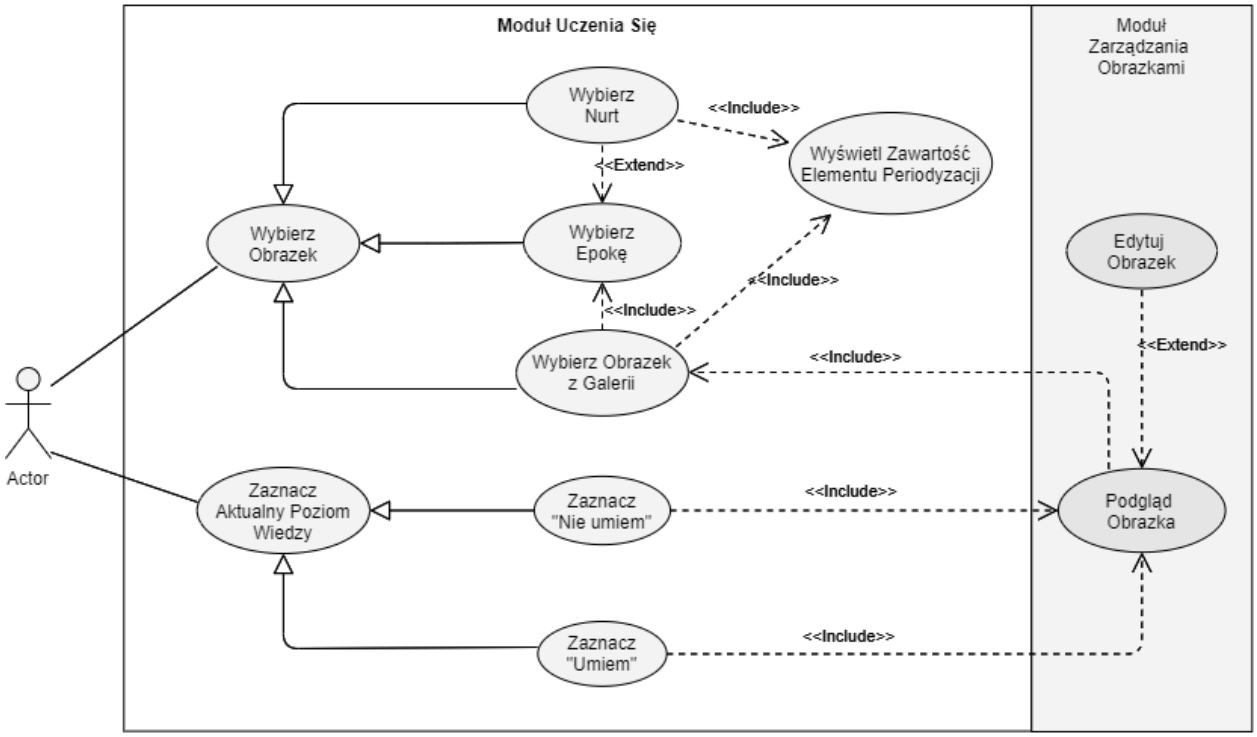
Rysunki 4 – 7 prezentują diagramy przypadków użycia dotyczące kolejno wszystkich Modułów opisanych w podrozdziale 2.1.1. Wszystkie diagramy wykonano przy pomocy narzędzia Visual Paradigm [27].



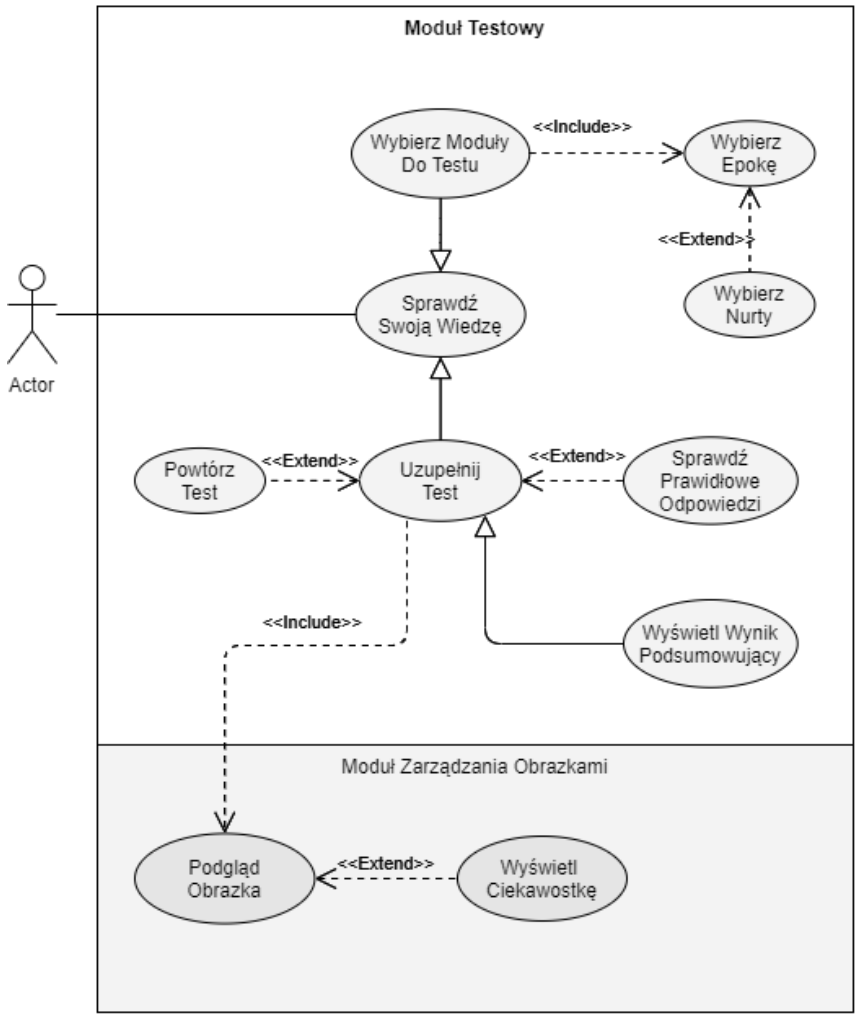
Rys. 4. Diagram przypadków użycia – Moduł Zarządzania Periodyzacją [opracowanie własne]



Rys. 5. Diagram przypadków użycia – Moduł Zarządzania Obrazkami [opracowanie własne]



Rys. 6. Diagram przypadków użycia – Moduł Uczenia Się [opracowanie własne]



Rys. 7. Diagram przypadków użycia – Moduł Testowy [opracowanie własne]

Jak widać na załączonych powyżej diagramach, żaden z modułów nie funkcjonuje całkowicie samodzielnie w odrębnej przestrzeni, natomiast znacząco przenikają się już na poziomie logicznym aplikacji.

2.2. Wymagania niefunkcjonalne

Wymagania niefunkcjonalne opisują funkcje jakościowe systemu, czyli jego właściwości (czas reakcji, niezawodność, zajętość pamięci, wydajność) i ograniczenia (czasowe, dotyczące procesu tworzenia, standardy) [26].

Wymagania niefunkcjonalne „wynikają z: potrzeb Użytkownika, ograniczeń budżetowych, strategii firmy, konieczności współpracy z innymi systemami (sprzętu lub oprogramowania), czynników zewnętrznych jak przepisy o bezpieczeństwie, ustawy chroniące prywatność," i tym podobnych czynników [26].

Zestawienie wymagań niefunkcjonalnych dotyczących aplikacji HaSZ prezentuje się następująco:

- Aplikacja HaSZ ma działać na urządzeniach mobilnych wyposażonych w system Android, API 29 i nowsze.
- Aplikacja ma posiadać wbudowaną bazę danych zawierającą informacje o wszystkich opisanych przez nią plikach graficznych, w szczególności dotyczy to ścieżki do pliku graficznego przypisanego do konkretnego obiektu Obrazka.
- Aplikacja musi mieć dostęp do pamięci urządzenia, na którym się znajduje.
- W przypadku pobierania dedykowanych pakietów, pliki graficzne Obrazków mają zostać umieszczone w folderze aplikacji, jednak w przypadku wszystkich Obrazków dodawanych przez Użytkownika aplikacja potrzebuje stałego dostępu do wybranych przez Użytkownika plików.
- Aplikacja nie może kopiować plików graficznych Użytkownika ani zmieniać ich w żaden sposób. Jedyne pliki, które Aplikacja może edytować, to jej własne pliki, w szczególności należące do wbudowanej bazy danych.
- Aplikacja ma za pomocą komunikatów tekstowych informować Użytkownika o aktualnie przeprowadzanych operacjach oraz ich wyniku.
- Aplikacja musi posiadać responsywny interfejs, automatycznie dostosowujący się do różnej wielkości ekranów urządzeń końcowych.
- Aplikacja musi działać przy wertykalnym położeniu ekranu, nie musi dostosowywać się do położenia horyzontalnego.

- Aplikacja powinna działać szybko i sprawnie, nie powinna się zacinać. Nie może też blokować interfejsu Użytkownika w czasie, gdy wykonuje obliczenia.
- Aplikacja musi posiadać interfejs o stonowanych, spokojnych kolorach. Aplikacja nie może zawierać elementów o zwiększonej, rażącej barwie, ani elementów dostarczających zbyt wiele bodźców wzrokowych.
- Aplikacja nie może zawierać żadnych własnych generatorów bodźców dźwiękowych. Dopuszczalne są jedynie domyślne dźwięki systemowe, takie jak dźwięk wciśnięcia przycisku.
- Aplikacja nie może posiadać funkcjonalności, które nie są niezbędne do realizacji wymagań funkcjonalnych.
- Aplikacja nie powinna posiadać systemu powiadomień ani innych systemów posiadających potencjał niepokojenia Użytkownika i dostarczania mu zbędnych sygnałów / informacji.
- Aplikacja ma zawierać jedynie niezbędne informacje. Nie może wyświetlać okien (w szczególności kolorowych) innych niż te zawierające niezbędne komunikaty.
- Aplikacja musi funkcjonować w pełni sprawnie bez dostępu do Internetu (czyli w trybie *offline*). Dostęp do sieci ma być wymagany jedynie w celu pobrania samej aplikacji oraz pobierania gotowych modułów.

2.3. Architektura

„Architektura oprogramowania – jest to podstawowa organizacja systemu wraz z jego komponentami, wzajemnymi powiązaniami, środowiskiem pracy i regułami ustanawiającymi sposób jej budowy i rozwoju" [27].

Wybór wzorca architektonicznego dostosowanego do rodzaju oraz potrzeb aplikacji pomaga zarówno w procesie projektowania i tworzenia samego oprogramowania, jak i w późniejszych procesach utrzymania aplikacji oraz jej rozwoju.

2.3.1. Model MVVM

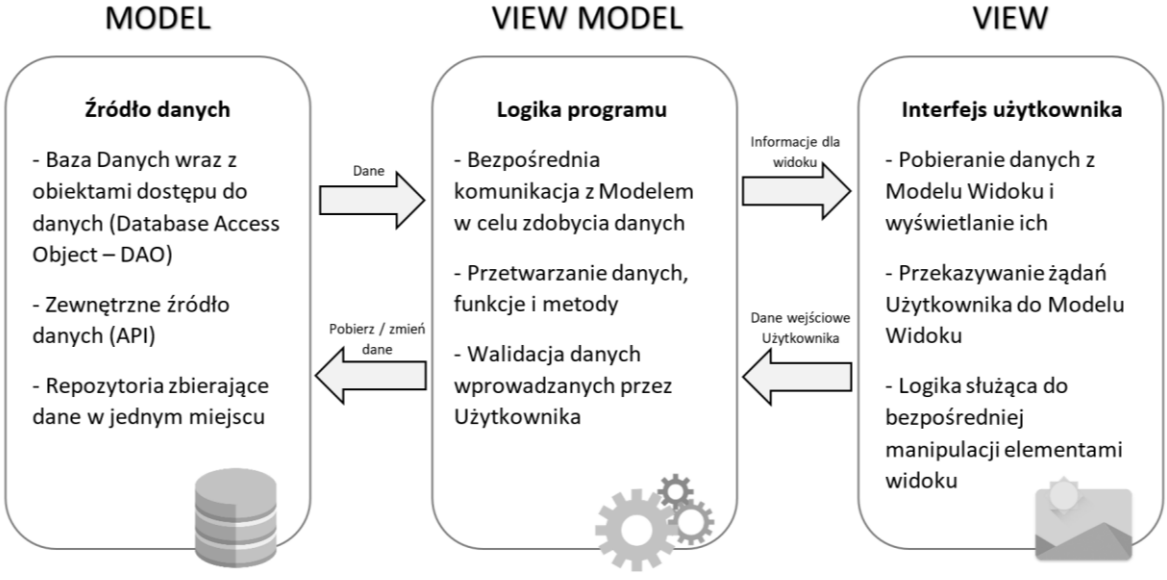
Aplikacja HaSZ oparta została o architektoniczny wzorzec o nazwie „*Model-View-ViewModel*", w skrócie MVVM, którego graficzną reprezentację zawiera rys. 8. Zakłada on podzielenie aplikacji na trzy główne części:

- **Model** – część zawierająca dane i zarządzająca nimi. Oznacza to dane znajdujące się w bazie danych, ale również dane pochodzące z zewnętrznego API udostępnianego na przykład poprzez Internet. Ta część przetwarza wszystkie ważne dane związane z aplikacją, a więc to tutaj odbywa się właściwy odczyt i zapis danych. Część ta zapewnia reszcie aplikacji zestaw komend, które pozwalają na wszystkie operacje związane z zarządzaniem danymi, a zatem zawiera logikę biznesową aplikacji. To ważne, aby wszystkie komponenty należące do innych części nie musiały w żaden sposób definiować operacji bazodanowych, a mogły jedynie wywołać konkretną komendę, która wykonuje wszystkie potrzebne kroki w ramach Modelu. Dzięki temu wszystko znajdujące się ponad tą częścią nie musi już dbać o pochodzenie danych ani ich format.

- **Model Widoku** (***ViewModel***) – część programu odpowiedzialna za przygotowanie i zarządzanie danymi przeznaczonymi dla powiązanego komponentu widoku, takiego jak Fragment lub Aktywność. Umożliwia też komunikację powiązanego komponentu z resztą aplikacji, czyli Modelem i serwisami. To części *ViewModel* znajdują się wszystkie funkcje i algorytmy spełniające wymagania funkcjonalne aplikacji, to ta część decyduje o tym, co zostanie wyświetlone Użytkownikom, a także co stanie się, gdy Użytkownik wciśnie przycisk lub przesunie palcem po fragmencie ekranu. Ta część sprawdza też poprawność danych wprowadzonych przez Użytkownika i używa ich do dalszych obliczeń. Pobiera dane z części Modelowej i je przetwarza, lecz nie zawiera żadnej logiki zawierającej bezpośrednie operacje na danych, nie wie też, skąd owe dane pochodzą – otrzymuje jedynie gotowy

zestaw komend z Modelu, dzięki którym może poprosić Model o wykonanie potrzebnych obliczeń i udostępnienie gotowych obiektów zawierających wymagane dane. Otrzymane obiekty mają zawsze jeden, ustalony format, na którym Model Widoku następnie pracuje w celu wypełnienia wszystkich funkcjonalnych wymagań stawianych aplikacji.

- **Widok** (***View***) – część zajmująca się UI (*User Interface*, interfejs Użytkownika), czyli tym, co Użytkownik widzi i z czym przeprowadza interakcje. W modelu MVVM ta część nie wykonuje żadnych obliczeń związanych z przetwarzaniem danych. Jej zadaniem jest wyświetlanie interfejsu Użytkownikowi zgodnie z tym, co przekazuje jej Model Widoku oraz przekazywanie interakcji Użytkownika temuż Modelowi. Jedyne obliczenia, jakie są tu zawarte, dotyczą bezpośrednio komponentów zawartych w interfejsie Użytkownika, takich jak przyciski, pola wyświetlające tekst i obrazki, pola pozwalające na wprowadzanie tekstu, etc. Widok nie posiada informacji o pochodzeniu danych i nie musi ich zawierać, nie musi też przetwarzać danych, ani dbać o ich przechowywanie. Ściśle współpracuje z Modelem Widoku, otrzymuje od niego gotowe polecenia i przekazuje mu wszelkie interakcje, po czym oczekuje na kolejne instrukcje dotyczące aktualizacji panelów wyświetlanych Użytkownikowi.



Rys. 8. Graficzna reprezentacja modelu architektonicznego MVVM [opracowanie własne]

2.3.2. Architektura wykorzystana w implementacji aplikacji HaSZ

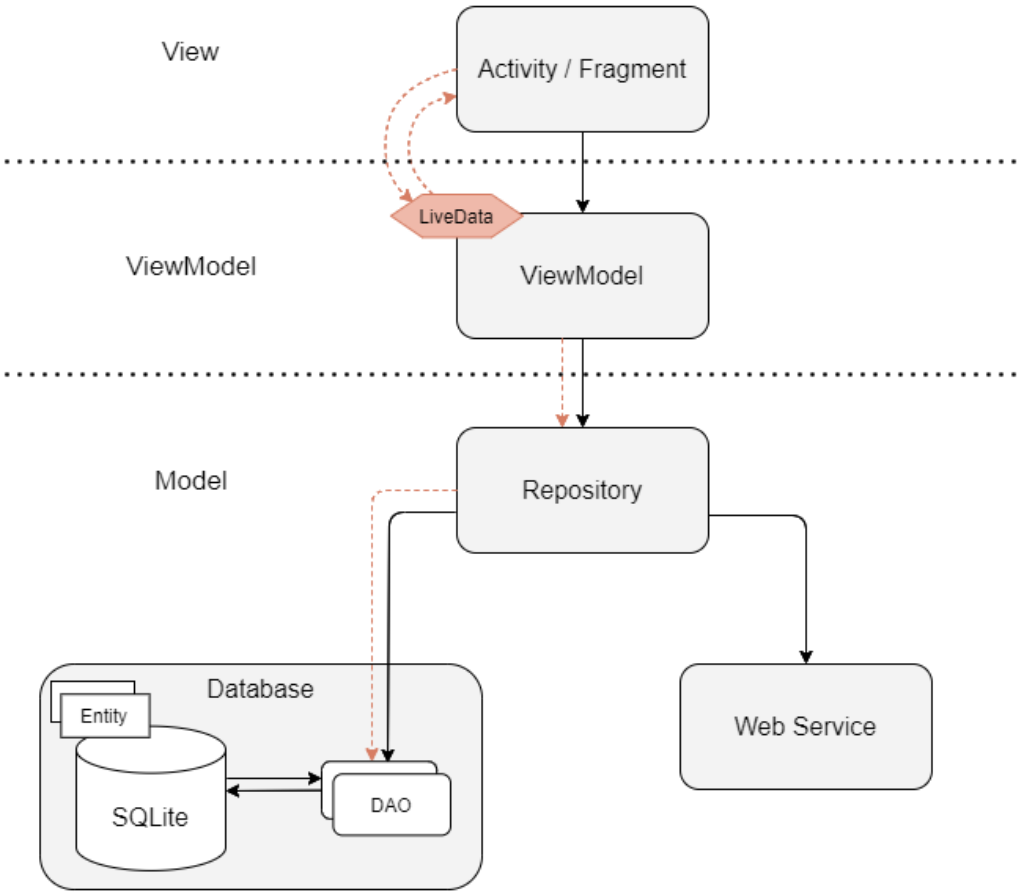
Aplikacja HaSZ została zaprojektowana i zrealizowana z wykorzystaniem architektury MVVM, jednak domyślna struktura tej architektury została dodatkowo wzbogacona o dodatkową warstwę umieszczoną w części Modelu składającą się z szeregu Repozytoriów. Służą one za „*wrapper classes*", czyli za klasy opakowujące. Stanowią swego rodzaju nakładki na korespondujące obiekty dostępu do danych (*Data Access Objects*, DAOs) i zapewniają dzięki temu dodatkową warstwę abstrakcji między Modelem, a Modelem Widoku.

Repozytoria mają dwa zadania:

- Pierwszym z nich jest zapewnienie, aby wszelkie operacje bazujące na danych wykonywane były w wątkach w tle, dzięki czemu główny wątek aplikacji nigdy nie jest obciążony ciężkimi i długotrwałymi operacjami bazodanowymi (takimi jak zapytania odczytu / zapisu w bazie danych).

- Drugim zadaniem jest natomiast dostarczenie spójnego zestawu komend Modelowi Widoku. W takiej sytuacji Model Widoku nie ma żadnego bezpośredniego połączenia z obiektami dostępu do danych, ani tym bardziej z samą bazą danych. Bazuje jedynie na tym, co udostępnia mu Repozytorium powiązane z typem używanego obiektu. Model Widoku może wykonywać jedynie te operacje związane z modelową częścią programu, które są zawarte w odpowiednim Repozytorium. Nie musi też w żaden sposób przejmować się formatem uzyskiwanych danych, gdyż repozytorium dba o to, aby obiekty danego typu zawsze wyglądały tak samo i udostępniały identyczne funkcjonalności.

Diagram zawarty w rys. 9 prezentuje schematyczne przedstawienie modelu architektonicznego MVVM wzbogaconego o warstwę Repozytoriów. Zawarty jest w nim również komponent o nazwie *LiveData* służący do obserwacji aktualnego stanu danych, dzięki czemu elementy aplikacji od nich zależne mogą na bieżąco reagować na zmiany w owych danych.



Rys. 9. Diagram prezentujący architekturę użytą w projekcie - MVVM wzbogaconą o warstwę repozytoriów przynależącą do części modelowej [opracowanie własne przy pomocy oprogramowania Visual Paradigm]

2.4. Użyte rozwiązania

Oprócz nakreślenia wymogów funkcjonalnych i niefunkcjonalnych projektu oraz podjęcia decyzji odnośnie architektury, która zostanie użyta do jego realizacji, ważnym punktem jest dobór technologii, przy pomocy których aplikacja zostanie zrealizowana oraz komponentów, które wspomogą jej implementację.

Dynamiczny rozwój sektora wytwarzania oprogramowania w dużej mierze oparty jest na udostępnianiu i następnie wykorzystywaniu serii gotowych rozwiązań. Dzięki temu deweloperzy nie są zmuszeni do tracenia czasu na powtarzaniu tych samych linijek kodu i tych samych błędów wciąż na nowo, przy każdym kolejnym projekcie. Gotowe rozwiązania udostępniane są zarówno przez firmy i korporacje, jak i przez szeregowych programistów, którzy dzielą się wymyślonym przez siebie kodem, aby oszczędzić czasu i pracy kolegom i koleżankom po fachu. Część rozwiązań jest oczywiście płatna i wymaga zakupu lub subskrypcji, jednak wiele technologii, bibliotek i repozytoriów udostępniana jest całkowicie za darmo, szczególnie jeśli ich wykorzystanie nie niesie ze sobą materialnych zysków.

Aplikacja HaSZ również korzysta z udostępnionych bibliotek i repozytoriów, wykorzystuje też dostępne technologie. Dzięki temu jest mniej podatna na błędy związane z implementacją problematycznych i czasochłonnych rozwiązań, których samodzielne tworzenie i testowanie wymagałoby nieraz całych lat pracy oraz bardzo specjalistycznej wiedzy dotyczącej wąskich dziedzin.

2.4.1. Technologie

Technologie dostępne dla deweloperów aplikacji mobilnych zawierają języki oprogramowania, które można wykorzystać w pisaniu aplikacji oraz środowisko służące do kompilacji aplikacji.

- **wybór zintegrowanego środowiska programistycznego (IDE)**

Aplikacje mobilne można tworzyć za pomocą środowisk takich jak Eclipse, DroidScript, czy Visual Studio, a bardziej rozbudowane aplikacje można oprzeć na silnikach, takich jak Unreal Engine. Jednak oficjalnym środowiskiem programistycznym dedykowanym implementacji aplikacji mobilnych na system Android jest Android Studio stworzone przez Google.

Android Studio zostało wyposażone we wszystkie elementy niezbędne do stworzenia wysokiej jakości aplikacji mobilnej, posiada też wiele usprawnień. Wśród nich najważniejsze to:

- Wbudowany inteligentny edytor kodu – który wspomaga programistę sugestiami dotyczącymi kodu. Dzięki jego propozycjom można pozbyć się niepotrzebnych fragmentów kodu i przekomponować potrzebne, co przekłada się na zmniejszenie ilości wykorzystywanych zasobów i lepsze działanie aplikacji.

- Emulator systemu Android – wyposażony w różne wersje systemu oraz emulowanych urządzeń. Pozwala na testowanie aplikacji na różnych typach urządzeń bez potrzeby fizycznego posiadania każdego z nich. Dodatkowo Android Studio pozwala na wprowadzanie części zmian w aplikacji bez potrzeby ponownego jej uruchamiania.

- Okna pomocnicze: Inspektor bazy danych, Menedżer zasobów, Przeglądarka plików urządzenia – znacząco wspomagają pisanie pracy oraz detekcję błędów podczas korzystania z emulatora.

W związku z powyższym, ze względu na oferowane wsparcie, do implementacji aplikacji HaSZ został wybrany program **Android Studio**.

- **wybór języka programowania**

Aplikacje dedykowane systemowi Android można pisać w zasadzie w większości najbardziej popularnych języków programowania, wystarczy dobrać środowisko, które będzie je wspierać. Nie jest problemem napisanie aplikacji w C#, C++, czy nawet przy wykorzystaniu Pythona, czy HTML [28]. Jednak dwoma oficjalnie wspieranymi językami są nieodmiennie Java i Kotlin.

Java od początku była oficjalnym językiem używanym przy budowaniu aplikacji mobilnych na system Android, lecz przez problemy prawne związane z przedsiębiorstwem Oracle (właścicielem Javy) Google postanowił kierować się w stronę języka na Javie opartego, lecz Javą niebędącego, którym jest Kotlin. W związku z tym, od roku 2019 to Kotlin jest oficjalnym językiem przeznaczonym do implementacji aplikacji mobilnych na system Android [29]. Nie mniej jednak, Java nadal posiada pełne wsparcie ze strony Google i nadal jest wykorzystywana przy budowaniu aplikacji mobilnych.

Wybór języka do realizacji niniejszej pracy nie był łatwy. Zarówno Java jak i Kotlin mają swoje zalety oraz wady. Ostatecznie Autorka zdecydowała się na wykorzystanie języka Java, z następujących powodów:

- Silne typowanie obiektów.

Choć za zaletę Kotlina uważane jest automatyczne typowanie obiektów, Autorka postrzega to nadal jako wadę. Typowanie obiektów czyni kod znacznie bardziej czytelnym i łatwiejszym do zrozumienia przez przyszłych programistów chcących z niego korzystać.

Ma to dwojakie znaczenie:

przede wszystkim pomaga uporządkować kod i prędzej wykryć błędy. Jasno określony typ obiektu ułatwia zauważenie ewentualnych nieścisłości, a rzutowanie nie jest na tyle kłopotliwe, by z jego powodu pozbawiać się tego ułatwienia. Ponadto, gdyby Autorka chciała pozbyć się konieczności typowania obiektów, istnieje wiele bibliotek umożliwiających uzyskanie tego efektu w Javie (na przykład biblioteka Lombok).

Drugim ważnym czynnikiem jest czytelność kodu wobec osoby, która ma z nim pierwszy kontakt. Aplikacja HaSZ jest projektem o wolnej licencji, co oznacza, że pełny kod aplikacji umieszczony zostanie w publicznie dostępnym repozytorium i każdy będzie mógł ów kod dowolnie modyfikować. Poprawiona czytelność ułatwi pracę z kodem zarówno nowym programistom, jak i samej Autorce, gdy postanowi ona wrócić do programu po dłuższej przerwie.

- Wykorzystanie wartości „*null*”.

Język Kotlin automatycznie czyni wszystkie zmienne obiektami, którym nie można przypisać wartości pustej, czyli „*null*”. To rozwiązanie zostało wprowadzone z powodu częstości występowania wyjątków prowadzących do awarii aplikacji, a powiązanych bezpośrednio z występującą w Javie możliwością tworzenia pustych (*null*) obiektów.

Tymczasem jest to wartość bardzo przydatna, gdyż może służyć za „darmową” flagę – sprawdzenie jej obecności może przekierować program na wykonywanie innego bloku kodu, a jednocześnie nie trzeba pamiętać o definiowaniu i uzupełnianiu dedykowanej flagi.

Ponadto, choć uciążliwe, wyjątki pustego wskaźnika (*null pointer exceptions*) są bardzo ważnym wyznacznikiem jakości pisanego kodu – w dobrze napisanym programie nie powinny występować i zawsze wskazują na błędy i niedopatrzenia programisty, co posiada dużą wartość edukacyjną i wspomagającą rozwój. Stąd przeświadczenie Autorki, iż z Kotlina powinni korzystać jedynie naprawdę doświadczeni programiści, którzy i tak ze wspomnianymi wyjątkami i awariami nie mają już wiele do czynienia i wiedzą jak pisać nieawaryjny, zadbany i dobrze przemyślany kod. W przeciwnym wypadku można niepotrzebnie utrwalić w sobie negatywne nawyki związane z pisaniem programów. Autorka uważa, że z wyjątków pustego wskaźnika może wynieść jeszcze sporo lekcji, zatem póki co nie zamierza z nich rezygnować.

- Kontrola.

Wybranie języka Java automatycznie oznacza większą kontrolę nad kodem. Oczywiście, oznacza również większą odpowiedzialność i potrzebę doglądania wielu szczegółów. Oba te czynniki pomagają w rozwoju i wzbogacają programistyczną wiedzę. Dodatkowa kontrola pozwala też na ograniczenie i tak już sporego kodu pobocznego, produkowanego automatycznie przy kompilacji.

- Objętość plików i czas kompilacji.

Co prawda wykorzystanie języka Kotlin oznacza mniej kodu podczas pisania aplikacji (w praktyce realna różnica widoczna jest jedynie w podstawowych klasach, do których i tak nie zagląda się często), ale jednocześnie przekłada się to na: większą ilość wymaganych bibliotek, dodatkowe tłumaczenie kodu na kod binarny Javy (Kotlin działa na JVM, maszynie wirtualnej Javy, więc i tak najpierw musi zostać przetłumaczony na czystą Javę), dłuższy czas budowania aplikacji.

2.4.2. Biblioteki

Poza podstawowym zestawem bibliotek umożliwiającym budowanie aplikacji mobilnych przeznaczonych na system Android, w implementacji aplikacji HaSZ zawarto następujące biblioteki:

- **Room persistence library** – biblioteka wspomagająca operacje bazodanowe z wykorzystaniem języka SQLite. Zapewnia adnotacje, dzięki którym część kodu generowana jest automatycznie – oznacza to mniejszą ilość błędów podczas pisania podstawowych zapytań bazodanowych. Ponadto weryfikuje zapytania napisane przez programistę już w czasie kompilacji programu, dzięki czemu wiele błędów można zlokalizować i skorygować zanim aplikacja w ogóle zacznie działać. Zmniejsza to znacznie ilość awarii aplikacji, które mogą pojawiać się z powodu drobnych błędów w zapytaniach. Room jest biblioteką polecaną oficjalnie dla implementacji aplikacji na system Android [30].

- **Bumptech Glide** – framework (platforma programistyczna) służący do wczytywania grafik i zarządzania ich plikami źródłowymi. Implementuje wydajne rozwiązania służące dekodowaniu plików graficznych (również w formacie .gif) i umieszczaniu ich w wyznaczonych miejscach w aplikacji przy jednoczesnym unikaniu wielu błędów i wyjątków [31].

- **Jetpack AndroidX Core** – biblioteka zawierająca klasy zapewniające wsparcie dla poprzednich wersji systemu Android. Pozwala na używanie API niezawartych w poprzednich wersjach systemu oraz API wykraczające poza aktualną platformę. Kompatybilna z urządzeniami wykorzystującymi system Android na poziomie API 14 i wyższym.

- **Jetpack AndroidX LifeCycle LiveData** – klasa dodatkowa, której zadaniem jest przechowywać zawsze aktualną wersję wskazanego obiektu (danych). Można ją obserwować z poziomu innych obiektów i komponentów. W przeciwieństwie do zwykłych obserwowalnych klas, *LiveData* została wyposażona w mechanizm podporządkowujący ją cyklowi życia komponentów, z którymi klasa ta współpracuje (takimi komponentami mogą być Aktywności, Fragmenty, Serwisy i tym podobne). Oznacza to, że klasa *LiveData* aktualizuje zawarte w niej dane tylko i wyłącznie wtedy, gdy powiązany z nią komponent znajduje się w stanie aktywnym. W przeciwnym wypadku klasa zawiesza działanie, aby nie wykorzystywać niepotrzebnie zasobów urządzenia. Ponadto, obiekty *LiveData* są automatycznie usuwane, gdy komponent, z którym są powiązane, przestaje istnieć. Zapobiega to wyciekom pamięci (*memory leaks*) [32].

- **Jetpack AndroidX LifeCycle ViewModel** – biblioteka dostarczająca klasę *ViewModel*, która, tak jak *LiveData*, jest sprzężona z cyklem życia komponentu, któremu służy. *ViewModel* powstaje w momencie tworzenia Aktywności lub Fragmentu i pozostaje aktywny tak długo, jak długo istnieje zakres z nim powiązany. Oznacza to niewrażliwość na zmiany konfiguracji (w przeciwieństwie do elementów widoku, które przy zmianie konfiguracji ulegają zniszczeniu wraz z zawartymi w nich danymi) [33].

- **Jetpack AndroidX Fragment** – pozwala na używanie w aplikacji funkcjonalności Fragmentów [34].

- **Jetpack AndroidX Legacy** – biblioteka zapewniająca wsteczną kompatybilność z poprzednimi wersjami systemu Android, dzięki czemu projekt może korzystać z API niedostępnych w starszych wersjach Androida [35].

2.4.3. Repozytoria

Dodatkowe wsparcie w zakresie estetyki interfejsu użytkownika zapewniły następujące repozytoria:

- **Makeramen RoundedImageView** – repozytorium pozwalające na zaokrąglanie kątów obrazków wyświetlanych w widokach aplikacji. Implementuje najlepsze aktualnie dostępne rozwiązania pozwalające na uzyskanie efektu zaokrąglonych brzegów obrazka, co oznacza minimalne możliwe wykorzystanie zasobów i brak zbędnego powielania grafik [36].

- **Xabaras Android RecyclerView SwipeDecorator** – repozytorium w postaci klasy użytkowej (*utility class*) zapewniające prosty sposób na dodawanie tła, ikon oraz etykiet elementom zawartym w *RecyclerView* podczas ich przesuwania na wybraną stronę (w prawo lub w lewo) [37]. Ma znaczenie estetyczne.

2.4.4. Klasy pomocnicze

- **„OnSwipeTouchListener.java” autorstwa Mirka Rusina** [38].

Klasę wykorzystuje się w celu nasłuchiwania gestów Użytkownika i określenia sposobu reakcji, gdy przesunie on palcem po ekranie w lewo lub w prawo.

- **„Permissions.java” autorstwa NightSkyDev** [39].

Klasa implementuje prosty, gotowy do użycia zestaw metod wyświetlających okno z tłumaczeniem powodu wystosowywania prośby o przyznanie dodatkowych uprawnień aplikacji, po którego wyłączeniu wysyłana jest prośba do Systemu o wyświetlenie okna pozwalającego na udzielenie wspomnianych pozwoleń. Klasa została zmodyfikowana na potrzeby projektu.

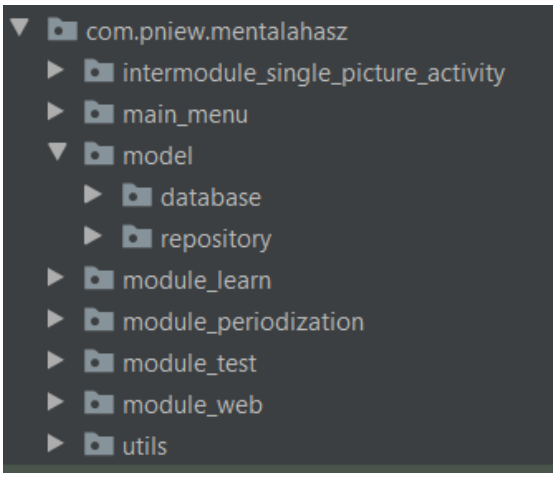
- **„RealPathUtil.java” autorstwa Tatocaster** [40].

Bardzo rozbudowana i przydatna klasa, oszczędzająca całe dni pracy. Jej wykorzystanie pozwala uzyskać ścieżkę do pliku na podstawie URI. Taką ścieżkę można następnie zapisać w bazie danych i przekazać na przykład bibliotece Glide, która na jej podstawie może dotrzeć do pliku, który ma za zadanie wyświetlić. Klasa wykazuje pełną wsteczną kompatybilność i działa na wszystkich API systemu Android.

3. Implementacja aplikacji

3.1. Opis ogólny

Aplikacja HaSZ składa się z pięciu modułów logicznych, których funkcjonalności przenikają się do tego stopnia, że implementacja ich jako faktycznie osobnych modułów okazała się pozbawiona podstaw. W związku z tym aplikacja zamiast na programistyczne moduły (które osiągnąć można poprzez modularyzację aplikacji przy pomocy narzędzia Gradle) podzielona została na niniejszy zestaw pakietów:



Rys. 10. Pakiety aplikacji HaSZ wydzielone katalogowo [opracowanie własne]

- Pakiet „model” zawiera wszystkie klasy wchodzące w skład części modelowej architektury MVVM.

- Pakiet „utils” przechowuje klasy pomocnicze, nie podlegające instancjonowaniu, zawierające szereg metod i pól statycznych wykorzystywanych na przestrzeni całej aplikacji, unifikująca ich wartości.

- Pakiet „main_menu” zawiera aktywność menu głównego, czyli pierwszego widoku, jaki zostaje wyświetlony po uruchomieniu aplikacji. Widok ten pozwala na uruchomienie każdego z modułów.

- Pakiet aktywności pojedynczego obrazka jest najważniejszym pakietem w aplikacji HaSZ – zawiera bowiem podstawę prawie wszystkich modułów.

- Pozostałe pakiety obejmują części widokowe (*View*) oraz przynależne im części modeli widoków (*ViewModel*) wraz z klasami pomocniczymi dedykowanymi danemu modułowi lub jednej z jego Aktywności (są to klasy Adapterów oraz ewentualne dodatkowe klasy pomocnicze).

3.2. Model

Część modelowa implementacji aplikacji HaSZ składa się z bazy danych oraz kompletu repozytoriów. Obie składowe współpracują ze sobą tworząc jednoznaczny i intuicyjny model, z którego może korzystać reszta aplikacji.

3.2.1. Baza danych i DAO

- **Bibliotera Room persistence**

Baza danych aplikacji HaSZ oparta jest na bibliotece Room *persistence library*. Biblioteka ta ułatwia tworzenie stabilnej bazy danych i samodzielnie tworzy relacje pomiędzy tabelami.

Korzystanie z biblioteki Room opiera się na używaniu adnotacji. Podstawowe komórki bazodanowe tworzy się poprzez umieszczenie adnotacji *@Entity* nad klasą, którą zamierza się przekształcić w tabelę. Następnie można przystąpić do deklaracji odpowiednich pól, które Room doda następnie w formie kolumn tabeli. Pole stanowiące klucz główny należy oznaczyć adnotacją *@PrimaryKey*. Jeśli istnieje wymaganie, by klucz główny był generowany automatycznie, w opcjach adnotacji należy umieścić instrukcję „*autoGenerate = true*", która przekaże to wymaganie bibliotece Room. Jeśli tabela posiada klucze obce kierujące do pól w innych tabelach, należy je oznaczyć przy pomocy adnotacji *@ForeignKey* oraz indeksów umieszczanych w opcjach adnotacji *@Entity*.

Należy pamiętać o dodaniu funkcji umożliwiających ustawienie oraz pobranie wartości dla każdej z kolumn (tak zwane „*gettery*" i „*settery*"), co umożliwi bibliotece Room wygenerowanie kodu niezbędnego do uzupełniania i aktualizacji tych pól w później tworzonych encjach bazodanowych.

W przypadku, gdy jest to potrzebne, można nadpisać metody:

- *toString(),* w której możemy zdecydować, co zostanie przekazane jako parametr wyjściowy w przypadku, gdy późniejsze działania będą zmierzać do odczytania obiektu opartego na tej klasie,

- *equals()*, używanej w momencie porównywania dwóch obiektów należących do tej klasy,

- *hashCode()* służącej do zwrócenia unikatowej wartości liczbowej dla każdego obiektu tej klasy.

Metody te generowane są automatycznie, lecz nadpisanie ich pozwala na przejęcie pełnej kontroli nad tym, w jaki sposób będzie przebiegało porównywanie obiektów oraz co wyświetli się po wywołaniu obiektu klasy.

- **Tabele w aplikacji HaSZ**

  Aplikacja HaSZ posiada cztery klasy stanowiące fundament bazy danych. Są to:

  - **Art Period** – Epoka sztuki,

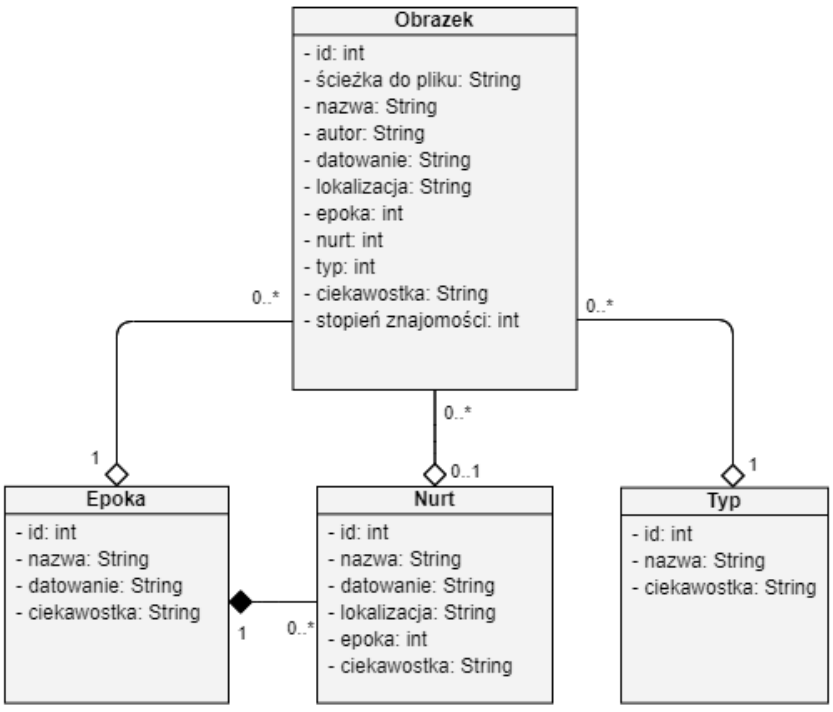
  - **Movement** – Nurt / okres sztuki,

  - **Type** – Typ sztuki,

  - **Picture** – Obrazek.

Nurty są przynależne epokom. Obrazek musi przynależeć do epoki oraz być jakiegoś typu, oraz może, ale nie musi, przynależeć do wybranego nurtu (oczywiście zawierającego się w wybranej epoce).

Poniższy diagram klas graficznie prezentuje wymienione zależności pomiędzy tabelami:



Rys. 11. Diagram klas aplikacji HaSZ [opracowanie własne przy pomocy oprogramowania Visual Paradigm]

Oprócz wymienionych klas, w aplikacji HaSZ używana jest dodatkowa klasa pomocnicza typu generycznego, bezpośrednio bazująca na tabelach, będąca tak zwanym widokiem bazy danych (*database view*). Widok bazy danych pozwala na enkapsulację wyniku zapytania do bazy danych w formę obiektu deklarowanej klasy. Zapytanie wykorzystane do stworzenia opisywanej klasy zaprezentowane zostało na rysunku nr 12.

Wykorzystana w projekcie klasa widoku bazy danych łączy oba elementy periodyzacyjne (Epoka, Nurt) oraz element opisowy (Typ) w jeden obiekt, zdolny przyjąć dane wszystkich typów elementów w formie obiektu pojedynczego typu. Umożliwia dzięki temu zastosowanie generycznego adaptera zdolnego przetworzyć wszystkie trzy typy obiektów jak jedną klasę.

```
("SELECT m.movementId AS id, m.movementName AS \"name\", m.movementDating AS \"dating\", " +
 "m.movementLocation AS \"location\", m.movementArtPeriod AS \"parent_art_period_id\"," +
 "a.artPeriodName AS \"in_period\",m.movementFunFact AS \"fun_fact\", \"Movement\" AS \"object_type\" FROM Movement m " +
 "JOIN ArtPeriod a ON artPeriodId = movementArtPeriod " +
 "UNION SELECT artPeriodId, artPeriodName, artPeriodDating, null, null, null, artPeriodFunFact, \"Art Period\" FROM ArtPeriod " +
 "UNION SELECT typeId, typeName, null,  null, null, null, typeFunFact, \"Type of Artwork\" FROM Type ORDER BY \"object_type\"")
```

Rys. 12. Zapytanie użyte do wygenerowania klasy unifikującej elementy periodyzacyjne [opracowanie własne]

- **Obiekty dostępu do danych – Data Access Objects, DAOs**

Obiekty dostępu do danych tworzą abstrakcyjną warstwę, poprzez którą enkapsulują źródło danych, oddzielając je od pozostałych warstw aplikacji. Tylko obiekty DAO mogą wykonywać operacje bezpośrednio na bazie danych, a zatem zapytania (*query*), w przypadku aplikacji HaSZ pisane w języku SQLite.

Jest to bardzo ważny element aplikacji, gdyż z tego poziomu można dokonać dosłownie dowolnych zmian w bazie danych (z destrukcyjnymi zmianami włącznie).

Aplikacja HaSZ zawiera zestaw obiektów DAO, po jednym dla każdej tabeli, jeden służący do obsługi dodatkowej wyżej opisanej klasy typu generycznego oraz jeden dodatkowy służący bibliotece Room do łączenia tabel odpowiednimi relacjami.

Biblioteka Room udostępnia zestaw adnotacji dedykowanych obiektom DAO, które pozwalają na automatyczne generowanie kodu przy dodawaniu, edycji, czy też usuwaniu encji. Jednak ze względu na dążenie do optymalizacji działania aplikacji (co oznacza możliwie najrzadsze wystosowywanie zapytań do bazy danych) Autorka stworzyła własny zestaw dedykowanych zapytań usprawniających działanie aplikacji.

Za przykład mogą tu posłużyć zapytania przedstawione na rysunku nr 13, edytujące jedynie jedno pole w krotce oraz usuwające krotkę jedynie przy pomocy numeru identyfikacyjnego. Zapytania generowane automatycznie przez bibliotekę Room cechują się wymogiem przekazania całego obiektu (z uzupełnionymi wszystkimi polami) nawet w celu usunięcia go, czy też edycji/uzupełnienia pojedynczego pola, co często powoduje dodatkowy kłopot związany z pozyskaniem wszystkich tych danych jedynie po to, by następnie je usunąć.

```
@Query("UPDATE Picture SET pictureFunFact = :pictureFunFact WHERE pictureId = :pictureId")
void setPictureFunFact(int pictureId, String pictureFunFact);

@Query("DELETE FROM Picture WHERE pictureId = :id")
void deletePicture(int id);
```

Rys. 13. Przykładowe zapytania stworzone na potrzeby aplikacji HaSZ [opracowanie własne]

3.2.2. Repozytoria

Repozytoria tworzą kolejną abstrakcyjną warstwę, tym razem nałożoną na obiekty typu DAO. W aplikacji HaSZ repozytoria pełnią funkcję rozbudowanych klas zarządzających modelem. Wywołują metody udostępnione przez obiekty DAO, ale robią to w wątkach pobocznych, dzięki czemu główny wątek nie jest obciążany długimi operacjami i nie zawiesza interfejsu graficznego podczas ich wykonywania.

Do zadań repozytoriów w aplikacji HaSZ należy też łączenie elementów przekazywanych z klas modułów w nowy obiekt, który następnie dodawany jest do bazy danych.

Natomiast w przypadku usuwania Obrazków repozytorium nadzorujące obiekty obrazków aktywuje też funkcję pozwalającą na usuwanie z folderu zasobów aplikacji plików pobranych z paczek, dzięki czemu usuwanie „obcych” dla Użytkownika obrazków (do których i tak nie ma dostępu z poziomu urządzenia) pociąga za sobą usunięcie zbędnych zasobów i zwolnienie tym samym części pamięci urządzenia.

Aplikacja HaSZ posiada pięć repozytoriów: po jednym dla każdej klasy typu *@Entity* oraz jedno dla klasy unifikującej elementy periodyzacyjne.

3.3. Widok i Model Widoku zawarte w pakietach modułów

Jak już wspomniano, aplikacja HaSZ podzielona została na pakiety zawierające logikę każdego z Modułów. Każdy z Modułów składa się z przynajmniej jednego elementu Aktywności, do której przypisany jest dedykowany Model Widoku oraz klas wspomagających. W kolejnych podrozdziałach zostaną wszystkie części aplikacji należące do warstw Widoku oraz Modelu Widoku.

3.4. Moduł Zarządzania Elementami Periodyzacji

„Elementy periodyzacji" to zbiorcze określenie trzech z obiektów tworzących tabele w bazie danych aplikacji HaSZ – Epoki, Nurtu oraz Typu dzieła sztuki. Choć Typ dzieła sztuki w istocie nie należy do elementów periodyzacji (periodyzacja jest umownym podzieleniem kontinuum czasowego, typ natomiast określa dziedzinę sztuki, do której dane dzieło przynależy), dla zachowania prostoty aplikacji oraz jej opisu wszystkie trzy elementy służące kategoryzowaniu określone zostały mianem periodyzacyjnych.

Moduł zarządzania tymi elementami podzielony jest na dwie dedykowane Aktywności:

- Aktywność wyświetlającą listę wszystkich elementów periodyzacyjnych,

- Aktywność wyświetlającą okno edycji oraz dodawania pojedynczego elementu.

Obie korzystają z klasy unifikującej wszystkie trzy typy obiektów i rozróżniają między nimi dopiero w momencie wyświetlenia odpowiednich elementów widoku.

- Aktywność listy elementów periodyzacyjnych

Wyświetla listę wszystkich elementów za pomocą widoku *RecyclerView*.

RecyclerView jest widokiem służącym do grupowania innych widoków. Od zwykłego widoku listy różni się tym, że potrafi w efektywny sposób zarządzać dużą ilością danych przeznaczonych do wyświetlenia oraz interakcji. Jest to spowodowane wykorzystaniem mechanizmu recyklingu – *RecyclerView* podczas generowania widoku listy tworzy poszczególne elementy, ale gdy Użytkownik przesuwa listę w poszukiwaniu kolejnych elementów, *RecyclerView* wykorzystuje ponownie elementy-widoki, które zostały przesunięte poza ekran. Dzięki temu nie tworzy ciągle nowych widoków dla kolejnych elementów,

co skutkuje mniejszym zużyciem zasobów urządzenia oraz szybszym czasem reakcji na wywoływane przez Użytkownika zmiany. Jednak *RecyclerView* nie jest w stanie samodzielnie wyświetlać danych.

Umieszczanie konkretnych widoków zawartych w *RecyclerView* należy do obowiązków Adaptera przypisanego do *RecyclerView*. Adapter jest łącznikiem pomiędzy źródłem danych przeznaczonych do wyświetlenia w formie listy oraz widokiem tejże listy. Jego zadaniem jest wygenerowanie gotowych widoków na podstawie zestawu danych, które otrzymuje.

W tym przypadku zestaw danych zawiera listę elementów periodyzacyjnych. Adapter sprawdza, do której klasy przynależy każdy z otrzymanych elementów i na tej podstawie generuje odpowiedni widok zawierający wszystkie dane dotyczące danego elementu. Dla Typu dzieła sztuki oznacza to widok zawierający jego nazwę, natomiast Epoka i Nurt zawierać będą odpowiednio większą ilość danych, jak choćby lokalizacja, czy datowanie. Adapter zajmuje się też podpisywaniem każdego widoku tak, aby Użytkownik nie miał wątpliwości co do rodzaju elementu, który aktualnie widzi.

Adapter otrzymuje swoje zestawy elementów od powiązanej z nim instancji klasy typu *LiveData*. Instancje tej klasy nieustannie obserwują przypisane im dane i w momencie, w którym zmienią się one na poziomie bazy danych, obserwator nałożony na obiekt *LiveData* zostaje powiadomiony i może uruchomić akcję powiązaną z wykryciem zmiany w zestawie obserwowanych elementów.

W tym przypadku obserwator *LiveData* przypisany jest do obiektu zawierającego zunifikowane dane z wszystkich trzech tabel dotyczących obiektów periodyzacji i jeśli zmiana zajdzie, nowy zestaw danych zostanie przekazany Adapterowi. Oczywiście dzięki byciu sprzężonym z cyklem życia aktywności, LiveData obserwuje i przekazuje dane jedynie wtedy, gdy aktywność jest aktualnie w stanie „rozpoczętym” (*started*) lub „wznowionym” (*resumed*). Eliminuje to błędy powiązane z cyklem życia komponentów.

Adapter wypełnia *RecyclerView* gotowymi widokami, natomiast *RecyclerView* wyświetla Użytkownikowi listę tych widoków, którą można przesuwać (*scroll*) oraz wchodzić w interakcje z poszczególnymi widokami.

Interakcje te obejmują:

- możliwość kliknięcia elementu na liście,

- możliwość przesunięcia elementu w lewo,

- możliwość przesunięcia elementu w prawo.

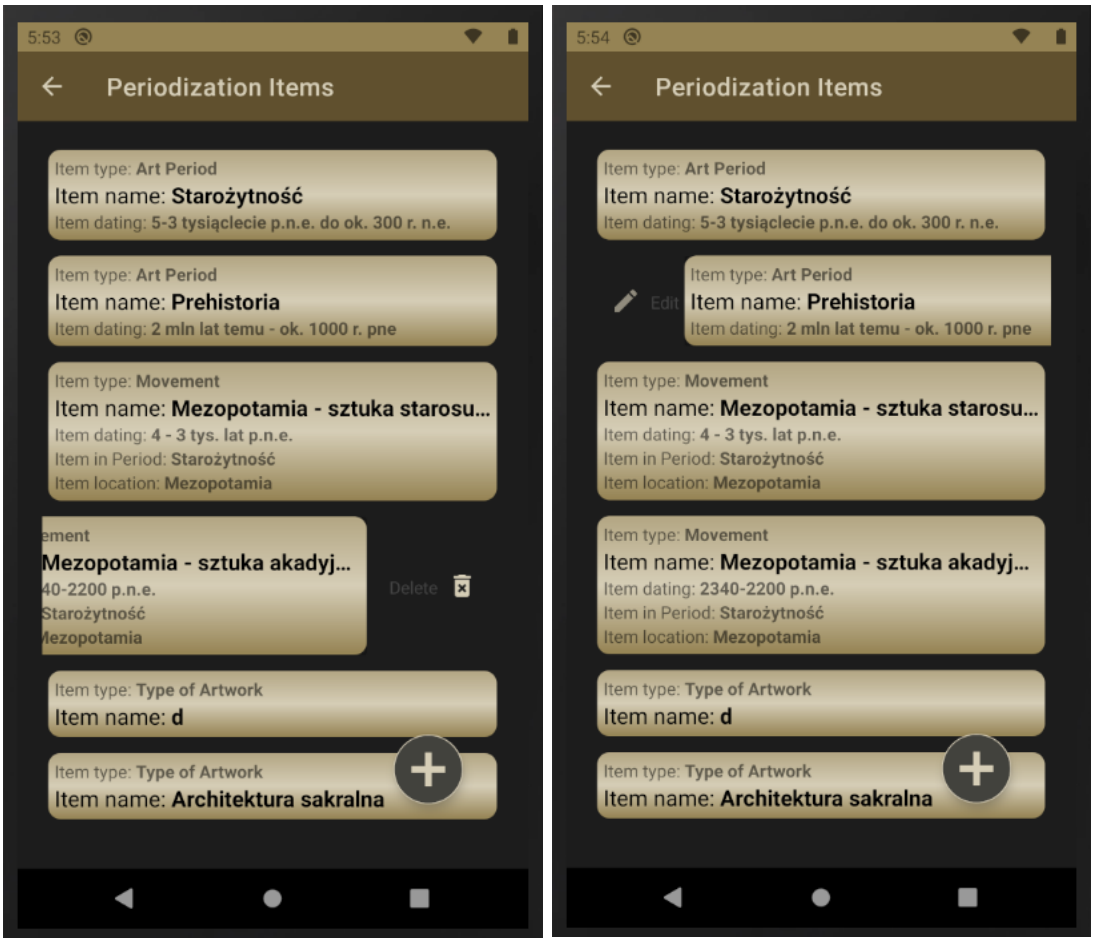
Kliknięcie w element oraz przesunięcie wybranego elementu w prawo powodują przejście do okna edycji elementu zawartego w drugiej aktywności, która wchodzi w skład tego Modułu.

Natomiast przesunięcie elementu w lewo zakłada chęć usunięcia owego elementu i powoduje wyświetlenie okna z prośbą potwierdzenia operacji. W przypadku zatwierdzenia operacji przez Użytkownika, element zostaje usunięty z bazy danych oraz, dzięki wykorzystaniu *LiveData*, z listy.

Repozytorium SwipeDecorator wspomaga tę aktywność poprzez umożliwienie umieszczenia w szybki i prosty sposób ikony oraz etykiety, które wyświetlają się podczas przesuwania elementów na boki, dzięki czemu Użytkownik może natychmiast zorientować się w typie operacji, którą uruchomi, kończąc przesunięcie. Dzięki temu Użytkownik może zrezygnować z wykonania operacji bez uruchamiania jej, jeśli jej nie potrzebuje.

Na dole ekranu znajduje się przycisk typu *Floating Action Button* zaopatrzony w odpowiednią ikonę, pozwalający na przejście do okna dodawania nowych elementów periodyzacyjnych.

Rysunek 14 przedstawia interfejs Aktywności listy elementów periodyzacyjnych. Zawiera on dwa zrzuty ekranu. Pierwszy zrzut prezentuje ikonę wyświetlaną podczas przesuwania wybranego elementu w lewo, drugi natomiast ikonę wyświetlaną podczas przesuwania wybranego elementu w prawo. Ikony te sugerują rodzaj operacji, jaki wykonany zostanie, jeśli przesunięcie nie zostanie wycofane. Na prezentowanych zrzutach ekranu widać również ilość pól z danymi na temat poszczególnych elementów oraz przycisk pozwalający na przeniesienie do Aktywności pozwalającej dodać nowy element periodyzacji (jest to okrągły przycisk ze znakiem „+” znajdujący się w dolnym prawym rogu ekranu).



Rys. 14. Widok listy elementów periodyzacji, Moduł Zarządzania Elementami Periodyzacji [opracowanie własne]

- Aktywność edycji oraz dodawania elementów periodyzacji

Ta aktywność zostaje wyświetlona w przypadkach, gdy:

- Użytkownik kliknął przycisk dodawania nowego elementu periodyzacji,

- Użytkownik użył możliwości dodania nowego elementu periodyzacji dostarczanej przez formularz dodawania/edycji obrazka dostępny w ramach Modułu Zarządzania Obrazkami,

- Użytkownik uruchomił Moduł Uczenia Się, ale nie posiada żadnych elementów periodyzacji, więc skorzystał z przycisku umożliwiającego dodanie takowych,

- Użytkownik kliknął w jeden z elementów wyświetlonych na liście,

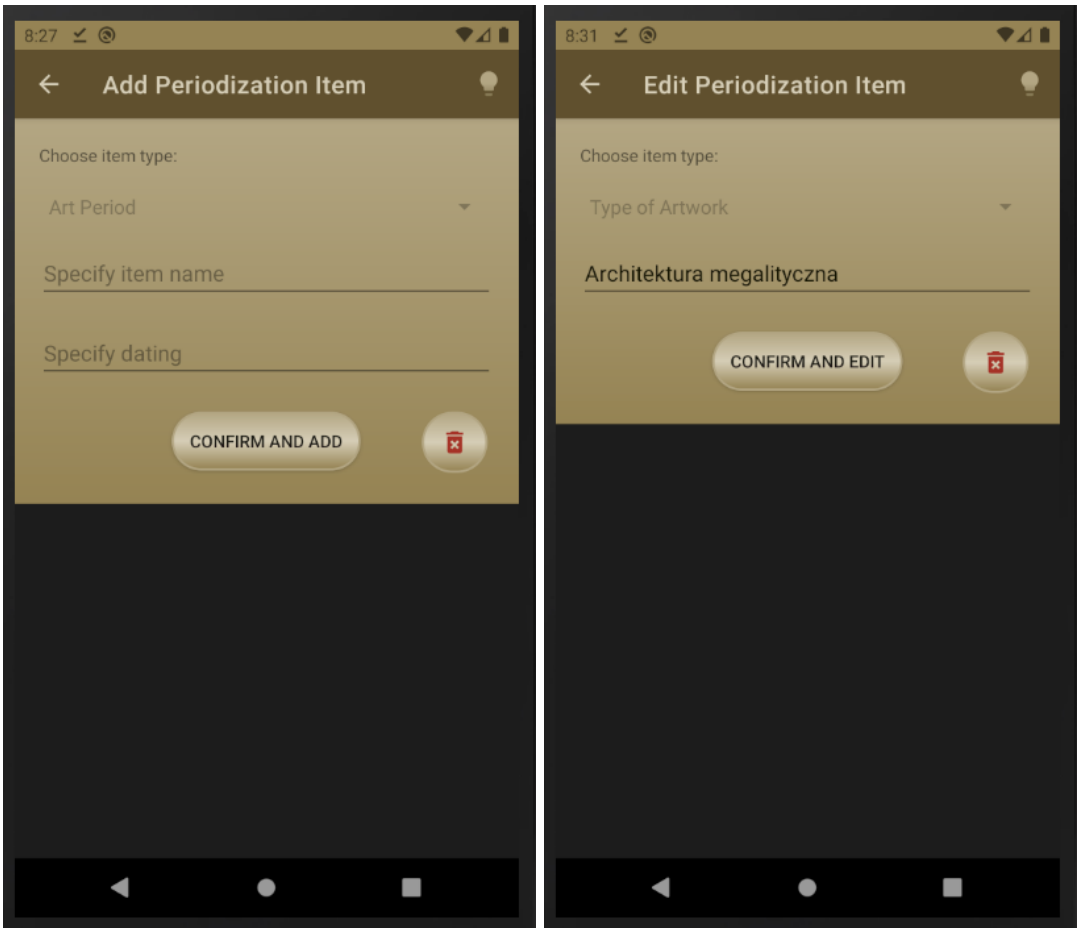
- Użytkownik przesunął jeden z elementów wyświetlonych na liście w prawo.

Aktywność wykorzystywana jest zarówno do dodawania, jak i edycji elementów periodyzacji, gdyż tworzenie osobnych aktywności do tych dwóch celów wiązałoby się z niepotrzebnym powielaniem kodu, co nie należy do prawidłowych praktyk programistycznych.

Aktywność zawiera formularz wyposażony w elementy niezbędne do dodawania oraz edycji wszystkich trzech typów elementów periodyzacyjnych, co oznacza obecność dwóch widoków *Spinnerów* (*Spinner* oznacza rozwijaną listę). Jeden służy do wyboru typu elementu, drugi wyświetlony jest jedynie przy edycji/dodawaniu Nurtów i służy do wyboru Epoki, do której dany Nurt ma przynależeć. Ponadto znajdują się tu trzy pola pozwalające na wpisywanie tekstu (jedno do wpisywania nazwy elementu, drugie przeznaczone do wpisania datowania, trzecie lokacji), przycisk zatwierdzający operację oraz przycisk anulujący operację.

Przejście do aktywności w celu edycji istniejącego elementu spowoduje przekazanie poprzez intencję informacji o edytowanym obiekcie. Dzięki temu aktywność nie musi powoływać własnej instancji klasy *LiveData* w celu zaobserwowania danych dotyczących edytowanego obiektu. Zamiast tego pobiera dane z przesłanej jej Intencji i uzupełnia nimi odpowiednie pola, które następnie Użytkownik może edytować.

W przypadku przejścia do aktywności w celu dodania nowego elementu pierwszy ze *Spinnerów* pozwoli na wybór dowolnego elementu periodyzacji. W przypadku edycji bądź przejścia do dodawania elementu z Modułu Zarządzania Obrazkami będzie on zablokowany na aktualnie wybranym typie elementu, co obrazują zrzuty ekranu zawarte w rysunku nr 15.

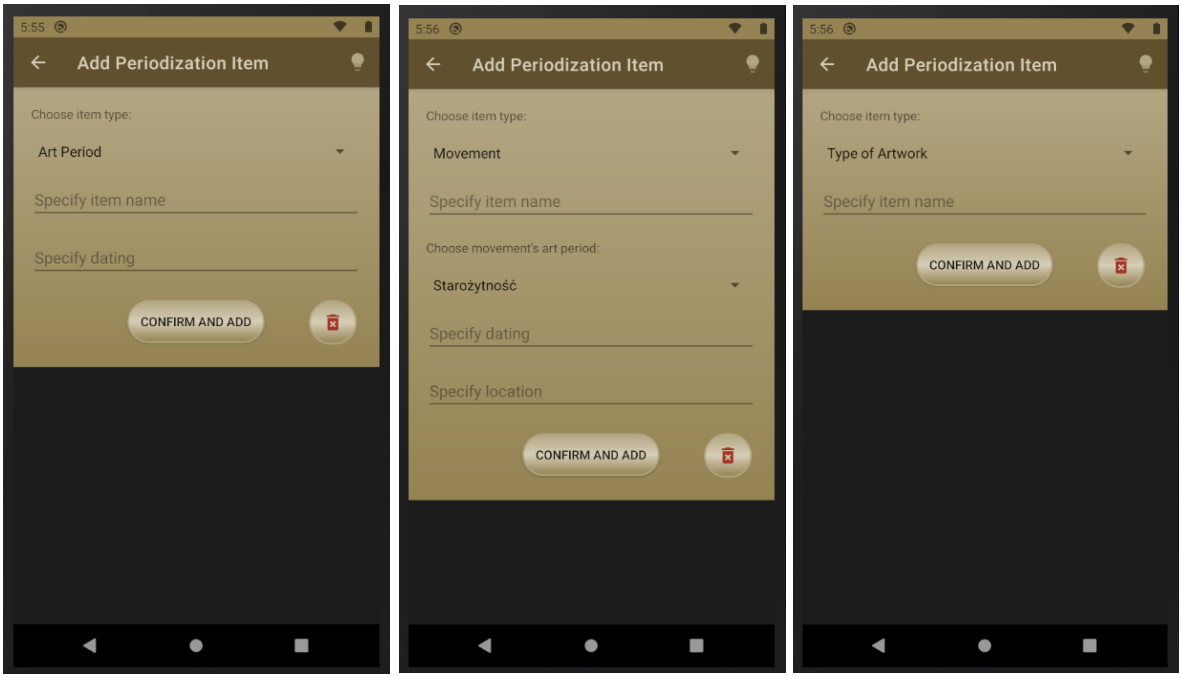


Rys. 15. Przykłady zablokowanego *Spinnera* w przypadku dodawania i edycji elementu periodyzacji [opracowanie własne]

Drugi *Spinner* w przypadku edycji nie jest blokowany, pozwala na zmianę przynależności Nurtu do innej Epoki. W przypadku przejścia do okna dodawania elementu z poziomu Modułu Uczenia Się jest natomiast zablokowany na pozycji Epoki, a której widoku okno zostało wywołane.

Zarówno drugi *Spinner*, jak i pola służące do wprowadzenia lokacji oraz datowania wyświetlają się jedynie w przypadku wybrania typu elementu, który potrzebuje ich w celu skompletowania danych (wszystkie trzy wariacje pola dodawania pokazane zostały na rysunku nr 16.

Górny pasek posiada ikonę pozwalającą na wyświetlenie i edycję ciekawostki dotyczącej aktualnie dodawanego/edytowanego elementu periodyzacji.



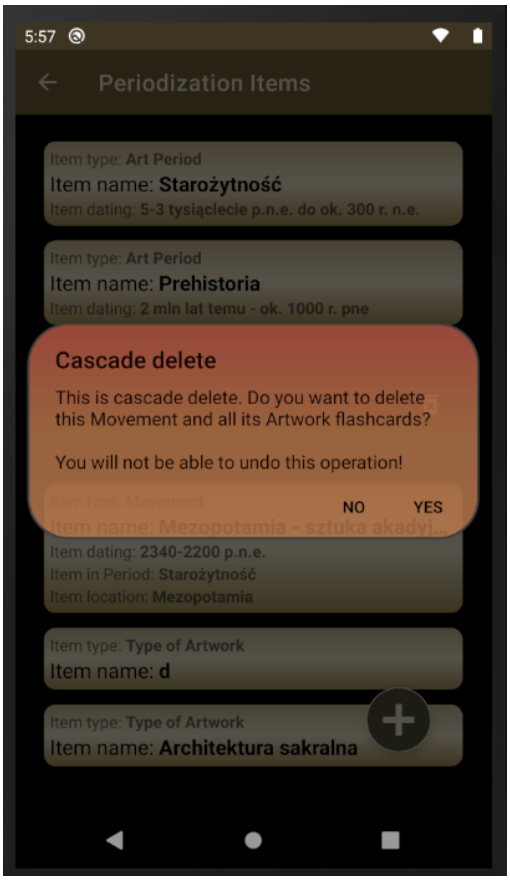
Rys. 16. Różne wersje okna dodawania w zależności od wybranego typu dodawanego elementu [opracowanie własne]

- Model Widoku Aktywności dodawania i edycji elementów periodyzacji

Aktywność powiązana jest z Modelem Widoku, który, zgodnie z założeniem, wykonuje wszelkie operacje związane z zarządzaniem danymi oraz przetworzeniem żądań Użytkownika.

Wstawianie oraz edycja elementów odbywa się za pomocą metody zawartej w Modelu Widoku. Metoda ta rozróżnia pomiędzy typem elementu, który ma obsłużyć, i w zależności od tego typu wykonuje inny fragment kodu. Wciśnięcie przez Użytkownika przycisku zatwierdzającego operację powoduje wysłanie do Modelu Widoku aktualnych danych znajdujących się w korespondujących polach wpisywania. Jeśli którekolwiek z pól jest puste, Model Widoku przekazuje aktywności treść komunikatu, który należy wyświetlić Użytkownikowi i nie wykonuje żądanej operacji. Dopiero, jeśli wszystkie wymagane pola zostaną uzupełnione, wykonana zostaje operacja wstawienia nowego obiektu do bazy danych, bądź też jego edycja.

Przycisk służący usuwaniu wybranego elementu oznaczony jest ikoną kosza, a po jego wybraniu, podobnie jak w poprzedniej aktywności, wyświetlone zostaje zapytanie proszące o potwierdzenie operacji (rys. 17). W przypadku zatwierdzenia, element zostaje usunięty z bazy danych.



Rys. 17. Komunikat ostrzegający Użytkownika przed kaskadowym usuwaniem obiektów [opracowanie własne]

Co istotne, następuje tu kaskadowe usuwanie obiektów z bazy danych. Dzieje się tak, ponieważ Nurt nie może istnieć bez nadrzędnej Epoki sztuki, natomiast Obrazek nie może istnieć bez przypisanych mu Epoki oraz Typu dzieła. W związku z powyższym usunięcie elementu periodyzacji, od którego zależne są inne elementy, bądź Obrazki, spowoduje usunięcie również wszystkich tych zależnych obiektów. Użytkownik jest o tym informowany w zapytaniu wyświetlanym przed każdą operacją usunięcia elementu periodyzacji.

3.5. Aktywność Pojedynczego obrazka oraz jej Model Widoku

Najważniejsza aktywność aplikacji HaSZ. Na niej opierają się wszystkie moduły związane z obrazkami, a zatem Moduł Zarządzania Obrazkami, Moduł Uczenia Się oraz Moduł Testowy.

Z powodu szerokiego pola zastosowań Aktywności, wymaga ona wielu danych przekazywanych jej przy powoływaniu jej do życia poprzez intencje. Są to informacje dotyczące:

- oczekiwań dotyczących Aktywności (mogą to być wartości oznaczające naukę, dodawanie Obrazka, edycję Obrazka, test),

- typu obiektu, który spowodował wywołanie (przy nauce będzie to odpowiednio galeria Epoki lub galeria Nurtu),

- numer identyfikacyjny elementu wywołującego (id Epoki, id Nurtu), a w przypadku, gdy wywołującym jest Nurt również numer identyfikacyjny Epoki, do której przynależy,

- numer identyfikacyjny Obrazka, który ma zostać wczytany,

- w przypadku, gdy przychodzimy do Aktywności w celu nauki lub przeprowadzenia testu również Tablica (*Array*) zawierająca numery identyfikacyjne wszystkich Obrazków, jakie przeznaczone są do wyświetlenia (oznacza to zawartość galerii, którą chcemy wyświetlić lub numery wszystkich Obrazków wyznaczonych do testu).

Aktywność wyposażona została w zestaw Fragmentów (umieszczanych w wyznaczonym ku temu miejscu, rys. 19) dedykowanych każdemu z wymienionych modułów:

- Fragment dodawania i edycji obrazka,

- Fragment służący do nauki obrazka,

- Fragment służący do przeprowadzania testu.

W zależności od potrzeb, aktywność wyświetla odpowiedni Fragment, dzięki czemu funkcjonalność wyświetlania pliku graficznego nie musi być wielokrotnie powielana. Model Widoku odpowiadający Aktywności pojedynczego obrazka przechowuje dane dotyczące aktualnie wyświetlanego obrazka, dzięki czemu ilość zapytań kierowanych do bazy danych może zostać maksymalnie zminimalizowana w przypadku przejścia pomiędzy

Fragmentami (przejścia takowe aktualnie występują między Fragmentem służącym do nauki oraz fragmentem edycji obrazka).

Aktywność posiada szereg metod pozwalających na działanie Fragmentów przynależących do odmiennych modułów. Ponadto zarządza wyświetlaniem i edycją Ciekawostek przypisanych wyświetlanym Obrazkom. Posiada również zestaw metod animujących zmiany w aktywności.

Aktywność posiada jeden widok wyświetlania Obrazka, który recyklinguje, podobnie jak *RecyclerView* wykorzystuje ponownie zawarte w nim widoki. Dzięki temu podczas wczytywania kolejnych Obrazków podczas nauki, czy też testu, nie zachodzi potrzeba tworzenia nowych Aktywności identycznych z istniejącą, nie zachodzi nawet potrzeba tworzenia nowych Widoków obrazu (*ImageView*) w ramach Aktywności, gdyż wykorzystuje ona cały czas ten sam widok. Jedynie Fragmenty ulegają wymianie na nowe. Pozwala to na znaczącą oszczędność zasobów urządzenia.

Zaimplementowane animacje dają natomiast wrażenia estetyczne przywodzące na myśl wymianę całych kart, dzięki czemu wrażenia Użytkownika aplikacji nie ulegają pogorszeniu pomimo zastosowania wymienionych oszczędności. Obrazki wczytywane są do widoku wyświetlania obrazu za pomocą biblioteki Glide (kod potrzebny do użycia biblioteki prezentuje rysunek nr 18), dzięki czemu prawdopodobieństwa wystąpienia nietypowych błędów związanych z wczytywaniem pliku graficznego maleje do minimum, Glide jest bowiem biblioteką rozwijaną od wielu lat, regularnie aktualizowaną i dzięki temu przeciwdziałającą większości wyjątków i błędów związanych z wczytywaniem plików graficznych do okna aplikacji [31].

```
Glide.with( activity: LearnShowAddActivity.this)
        .load(viewModel.getPath())
        .into(imageView);
```

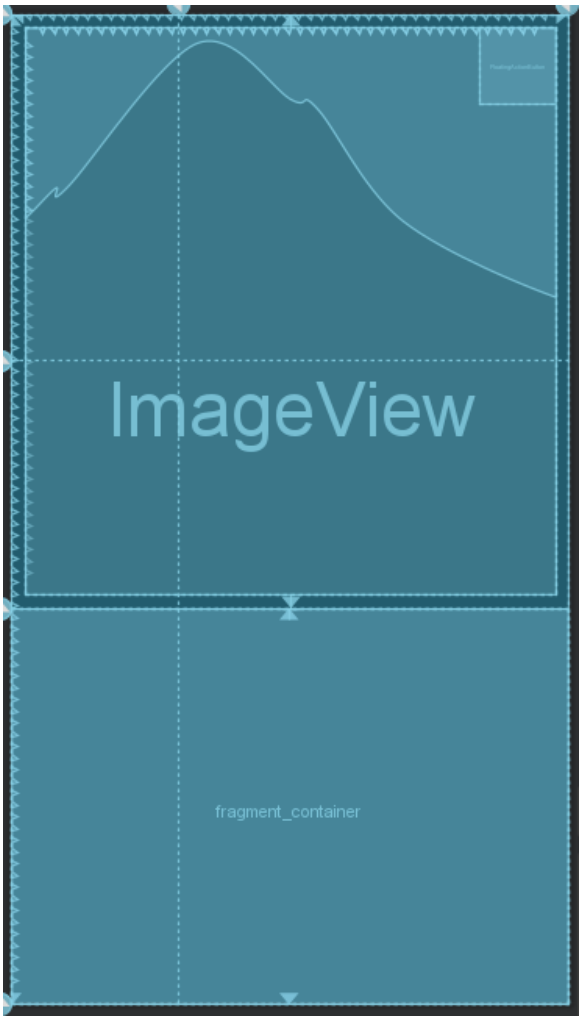
Rys. 18. Wykorzystanie biblioteki Glide do wczytywania plików graficznych w Aktywności Pojedynczego obrazka [opracowanie własne]

W przypadku, gdy Aktywność została wywołana w celu nauki lub sprawdzenia swojej wiedzy, pozwala ona na przechodzenie pomiędzy kolejnymi obrazkami z przekazanej jej Listy. Dla Modułu Uczenia Się jest to Lista zawierająca numery id wszystkich Obrazków znajdujących się w galerii aktualnie poznawanej Epoki / Nurtu, natomiast w przypadku testu Lista zawiera numery id wszystkich Obrazków wchodzących w jego skład. Lista ta nie pozostaje statyczna przez cały czas funkcjonowania aktywności.

W przypadku uczenia się Lista zostaje przetasowana za każdym razem, gdy Użytkownik dociera do jej końca, a następnie można rozpocząć naukę na nowo, tylko z inną kolejnością Obrazków. Zadaniem tego rozwiązania jest przeciwdziałanie uczeniu się na pamięć kolejności obrazków i uzależnianiu odpowiedzi od tej kolejności.

W przypadku testu natomiast tworzona jest dodatkowa, rezerwowa Lista, która pozostaje nietknięta przez cały czas trwania testu. Lista główna jest natomiast cały czas uszczuplana o każdy Obrazek, który został już w trakcie testu zatwierdzony. Gdy obrazki się skończą, wyświetlany zostaje wynik testu. Jeśli natomiast Użytkownik zechce powtórzyć test, Lista rezerwowa zostaje wykorzystana do odnowienia Listy głównej, dzięki czemu nie trzeba ponownie wczytywać numerów identyfikacyjnych Obrazków wchodzących w skład testu.

Obie Listy są oczywiście przechowywane i modyfikowane w Modelu Widoku. Oprócz tego wykonuje on operacje niezbędne do wczytania kolejnego lub poprzedniego elementu z Listy (Moduł Uczenia Się zakłada możliwość obustronnej nawigacji pomiędzy Obrazkami), przechowuje sumę punktów uzyskanych w teście, w razie konieczności aktualizuje Ciekawostkę dotyczącą aktualnie edytowanego obrazka. Bardziej szczegółowymi operacjami ograniczonymi do funkcjonalności konkretnych Modułów zajmują się Modele Widoku powiązane z poszczególnymi Fragmentami. Każdy z tych Fragmentów opisany zostanie dokładniej przy okazji przybliżania modułu, do którego logicznie przynależy.



Rys. 19. Rozkład elementów interfejsu użytkownika w Aktywności Pojedynczego obrazka [opracowanie własne]

3.6. Moduł Zarządzania Obrazkami

Moduł zarządzania Obrazkami obejmuje:

- dodawanie nowego obrazka,

- wyświetlanie wybranego obrazka,

- edycja istniejącego obrazka,

- usuwanie wybranego obrazka.

Technicznie rzecz biorąc, dodawanie Obrazków poprzez pobieranie gotowych zasobów również jest formą zarządzania Obrazkami, lecz wszystkie elementy powiązane z pobieraniem treści przynależą do Modułu Pobierania Zasobów, który został opisany osobno w podrozdziale 3.7.

Moduł służący do zarządzania Obrazkami (poza elementem służącym do pobierania Obrazków) opiera się całkowicie na przedstawionej w poprzednim podrozdziale Aktywności Pojedynczego obrazka.

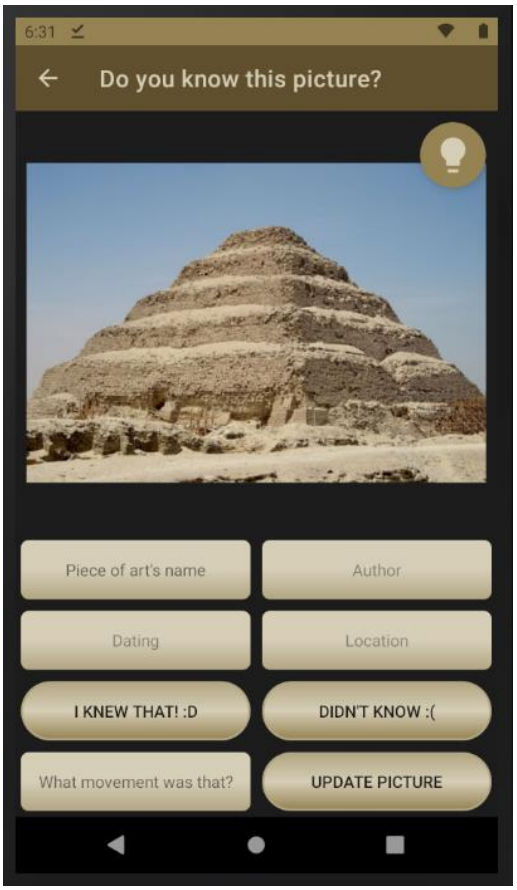
Do Modułu dostać się można na następujące sposoby:

- poprzez wciśnięcie przycisku „dodaj nową kartę” w Menu Głównym,

- za pomocą wciśnięcia przycisku z ikoną plusa w widoku galerii wybranej Epoki / Nurtu w Module Uczenia Się,

- poprzez wybranie opcji „edytuj obrazek” podczas wyświetlania wybranego obrazka w trakcie nauki w ramach Modułu Uczenia Się (rys. 20).

Każde wywołanie Aktywności Pojedynczego obrazka mającej zawierać Fragment dodawania lub edycji obrazka wymaga sprecyzowania tego celu w intencji uruchomienia Aktywności. Aktywność następnie przekazuje ów cel Fragmentowi w postaci pliku tobołka (*Bundle*).



Rys. 20. Widok Obrazka w ramach Modułu Uczenia Się. Dostępny jest tu przycisk pozwalający na przejście do edycji obrazka – „Update Picture” [opracowanie własne]

Klasami należącymi do Modułu są:

- Fragment Dodawania i Edycji obrazka (*AddEditFragment*)

- Widok Modelu Dodawania i Edycji obrazka (*AddEditViewModel*)

Dodatkowo posiada on też dedykowane metody oraz ich fragmenty w Aktywności Pojedynczego obrazka.

W związku z tym Moduł Zarządzania Obrazkami jako jedyny nie posiada własnego pakietu wyodrębnionego w strukturze katalogów projektu. Pod tym względem obie jego klasy znaleźć można w katalogu przeznaczonym na przechowywanie Fragmentów w pakiecie Aktywności Pojedynczego obrazka.

- Fragment Dodawania i Edycji obrazka

  Składa się z kolekcji widoków:

  - 4 widoki typu *EditText*,

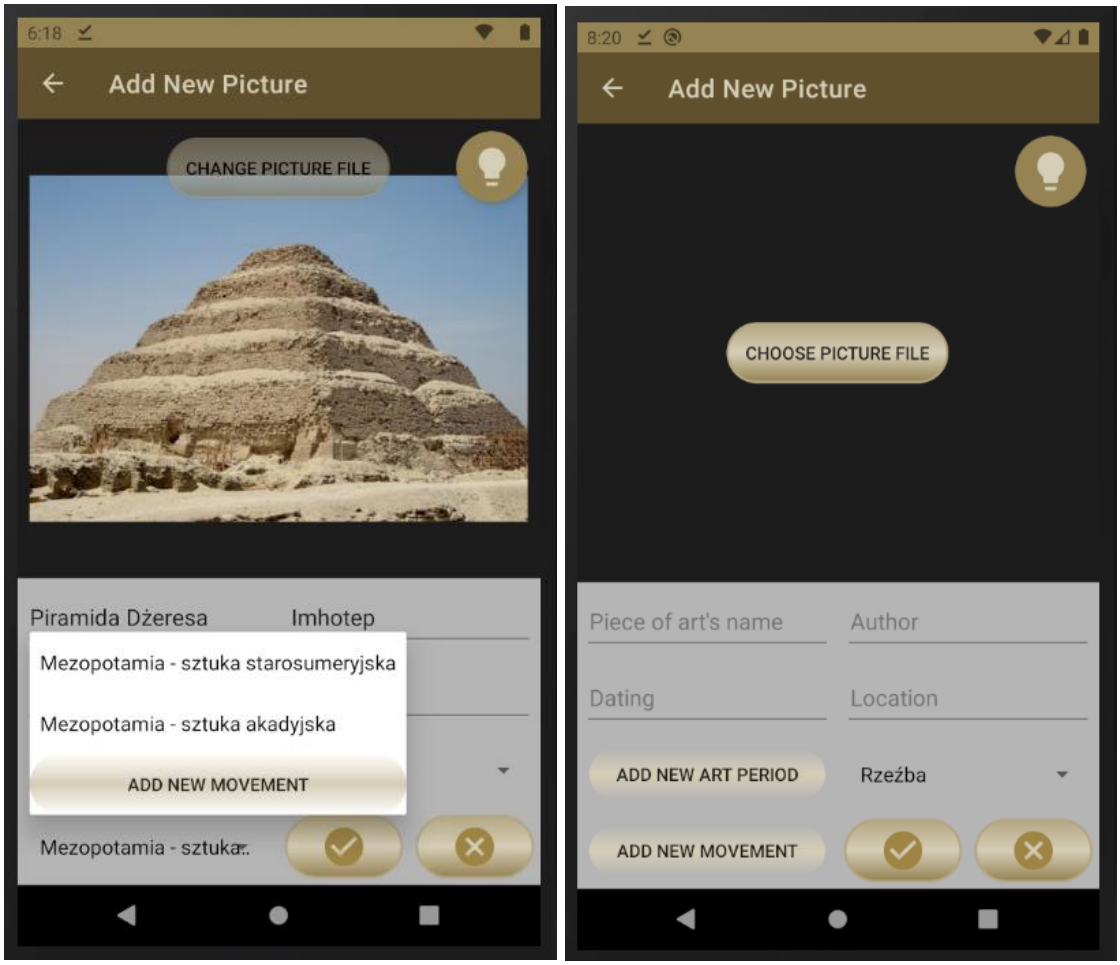
  - 3 widoki typu *Spinner*,

  - 2 przyciski (*Button*).

  Wszystkie widoki zawarto w elemencie pozwalającym na ich przesuwanie (*ScrollView*), dzięki czemu w trakcie wpisywania i wybierania wartości Użytkownik nie musi ukrywać klawiatury ekranowej w celu uzyskania możliwości wybrania lub podejrzenia kolejnego elementu.

  Widoki służące do edycji tekstu służą wprowadzaniu informacji dotyczących aktualnie dodawanego lub edytowanego obrazka. W przypadku uruchamiania Fragmentu w celu dodania nowego obrazka są puste, natomiast w przypadku edycji obrazka zostają wypełnione aktualnymi danymi edytowanego obrazka pobieranymi przy pomocy *LiveData*. Obserwator nałożony na obiekt *LiveData* zostaje usunięty po jednorazowym pobraniu danych dotyczących aktualnie edytowanego obrazka, gdyż jego stała obecność nie jest konieczna, natomiast wykorzystuje zasoby urządzenia. Wszystkie dane pobrane przy pomocy Obserwatora *LiveData* zostają natychmiast przekazane Modelowi Widoku (*AddEditViewModel*), gdyż Widok (w tym przypadku Fragment) nie powinien przechowywać danych niezwiązanych bezpośrednio z elementami widoku. Dopiero po przekazaniu wszystkich danych Modelowi Widoku następuje aktualizacja pól *EditText* zawartych we Fragmencie.

Widoki *Spinnerów* służą do wyświetlenia elementów periodyzacji, do których Użytkownik może przypisać Obrazek. *Spinnery* zawierają: listę Epok, listę Nurtów oraz listę Typów dzieła sztuki. Wszystkie wypełniane są danymi uzyskiwanymi przy pomocy dedykowanych Obserwatorów *LiveData*, które, w przeciwieństwie do powyżej opisanego obserwatora, nie zostają usunięte po pierwszym pobraniu danych. Powodem jest konieczność aktualizacji danych w przypadku, gdyby Użytkownik w trakcie dodawania lub edycji obrazka zdecydował się na dodanie nowego elementu periodyzacji. Jest to umożliwione dzięki dodaniu odpowiedniego przycisku na końcu list przekazywanych do Adapterów każdego ze *Spinnerów* (rys. 21).



Rys. 21. Przyciski pozwalające na przejście do ekranu dodawania nowego elementu periodyzacji bezpośrednio z ekranu dodawania / edycji obrazka [opracowanie własne]

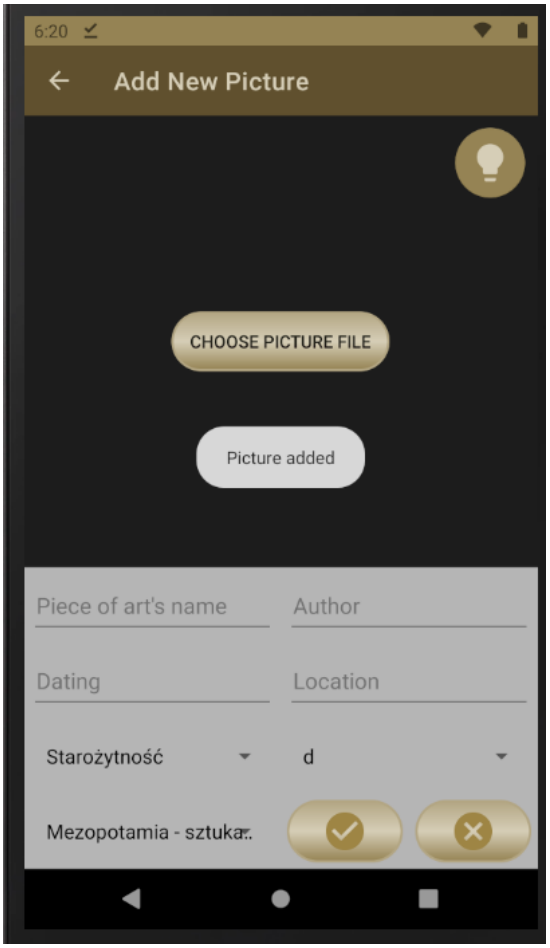
Adaptery *Spinnerów* są w tym celu modyfikowane podczas tworzenia ich instancji poprzez nadpisanie metod „*getDropDownView*” oraz „*getView*”. Metoda „*getDropDownView*” zwraca elementy, które wyświetlone zostają w liście, która pojawia się po kliknięciu w *Spinner*, gdy Adapter posiada w zestawie elementów więcej niż jeden obiekt. Metoda „*getView*” zostaje natomiast wywołana w celu uzyskania obiektu wyświetlanego jako pojedynczy element zwiniętego *Spinnera*. W obu metodach dodany został kod tworzący nowy przycisk w przypadku, gdy pozycja w liście Adaptera będzie wynosiła „*null*”, natomiast do listy przekazywanej Adapterowi zawsze jako ostatni dodawany jest pojedynczy pusty element, czyli właśnie element typu „*null*”. Zapewnia to obecność przycisku pozwalającego na dodanie nowego elementu periodyzacji na końcu listy w każdym ze *Spinnerów*, natomiast w przypadku, gdy nie istnieje jeszcze żaden element periodyzacyjny z danej kategorii, przycisk będzie widoczny bez potrzeby rozwijania *Spinnera*. Przyciski te przenoszą do opisanej w poprzednim podrozdziale Aktywności dodawania nowego elementu periodyzacji, przy czym z racji miejsca odniesienia, typ elementu jest od razu zaznaczony.

2 przyciski zawarte we Fragmencie służą kolejno do potwierdzenia operacji (ikona „Zatwierdź”, przycisk 1) oraz do anulowania operacji w przypadku dodawania nowego obrazka (ikona „Anuluj”, przycisk 2) lub usunięcia Obrazka w przypadku jego edycji (ikona „Usuń”, przycisk 2). Ikona przycisku anulowania / usuwania zmienia się w zależności od typu operacji przypisanej przyciskowi. Wybranie przycisku zatwierdzającego operację powoduje wywołanie metody „*saveUpdatePicture()*”, która pobiera z danych przekazanych przez Aktywność Pojedynczego obrazka ścieżkę do aktualnie wybranego pliku, jeśli zachodzi taka potrzebna (zachodzi zawsze w przypadku dodawania nowego obrazka oraz przy edycji obrazka w przypadku, gdy Użytkownik zdecydował o zmianie pliku graficznego przypisanego obrazkowi) i przekazuje ją swojemu Modelowi, a następnie w zależności od typu przeprowadzanej operacji aktywuje w tymże Modelu metodę „*savePicture()*” lub „*updatePicture()*”. Wynik metody jest przetwarzany, po czym:

- Fragment wyświetla komunikat z prośbą o uzupełnienie wszystkich pól, jeśli Użytkownik zażądał zapisania lub edycji obrazka nie uzupełniwszy wymaganych pól,

- Fragment wyświetla informację o poprawnym dodaniu Obrazka w postaci krótkiej wiadomości typu Toast, a następnie odświeża ekran dodawania (rys. 22), by umożliwić Użytkownikowi dodanie większej ilości obrazków, poprzez wywołanie metody „*refreshAddingPicture()*” w Aktywności Pojedynczego obrazka,

- Fragment wyświetla informację o tym, że obrazek został uaktualniony, po czym wywołuje w Aktywności metodę „*editFinished()*” aby powrócić do podglądu Obrazka w ramach Modułu Uczenia Się, lub metodę „*editFinished ChangedParticipation()*” w przypadku, gdy Użytkownik podczas edycji obrazka zmienił jego przynależność do Nurtu / Epoki, a zatem Obrazek zostaje usunięty z listy Obrazków przynależnych aktualnie wyświetlanej Epoki / Nurtu. W tej sytuacji następuje płynne przejście do następnego obrazka z listy i Użytkownik może kontynuować swoją naukę.

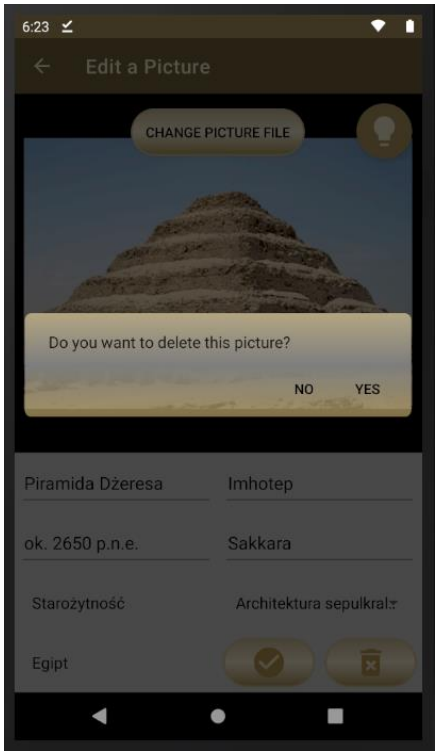


Rys. 22. Okno dodawania odświeżone po dodaniu nowego obrazka. Komunikat typu „*Toast*” informujący Użytkownika o pomyślnym wykonaniu polecenia dodania Obrazka do bazy danych [opracowanie własne]

Wybranie przycisku anulowania / usunięcia Obrazka powoduje natomiast:

- zamknięcie Fragmentu w przypadku dodawania Obrazka,

- wyświetlenia okna proszącego o potwierdzenie operacji usuwania w przypadku edycji obrazka (rys. 23). Potwierdzenie swoich zamiarów poprzez wybranie opcji potwierdzającej zawartej na dole okna spowoduje wywołanie metody „*deletePicture()*” zawartej w Modelu Fragmentu, a następnie wywołanie metody powodującej zamknięcie całej Aktywności Pojedynczego obrazka i powrót do galerii Epoki / Nurtu.



Rys. 23. Okno proszące o potwierdzenie operacji usuwania wybranego obrazka [opracowanie własne]

- **Model Widoku Dodawania i Edycji obrazka**

Przechowuje wszystkie dane dotyczące aktualnie edytowanego obrazka, posiada instancje wszystkich czterech Repozytoriów oraz instancje obiektów typu *LiveData*, które mogą być obserwowane z poziomu Fragmentu. Przechowuje pełną logikę metod służących do zapisywania, aktualizacji oraz usuwania Obrazków. W przypadku zapisywania i edycji obrazków sprawdza, czy wszystkie wymagane pola zostały wypełnione – jeśli nie są, wysyła informację widokowi Fragmentu, który następnie przekazuje ją Użytkownikowi.

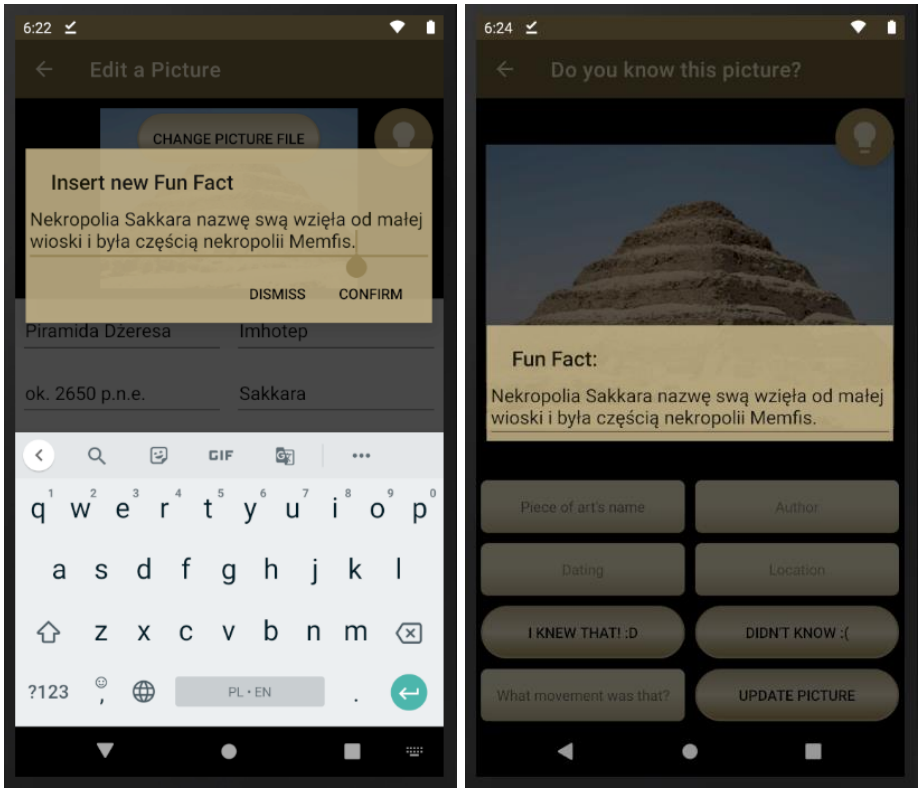
- **Aktywność Pojedynczego obrazka – obsługa Modułu Zarządzania Obrazkami**

Główna aktywność aplikacji HaSZ służy Modułowi zarządzania Obrazkiem na następujące sposoby:

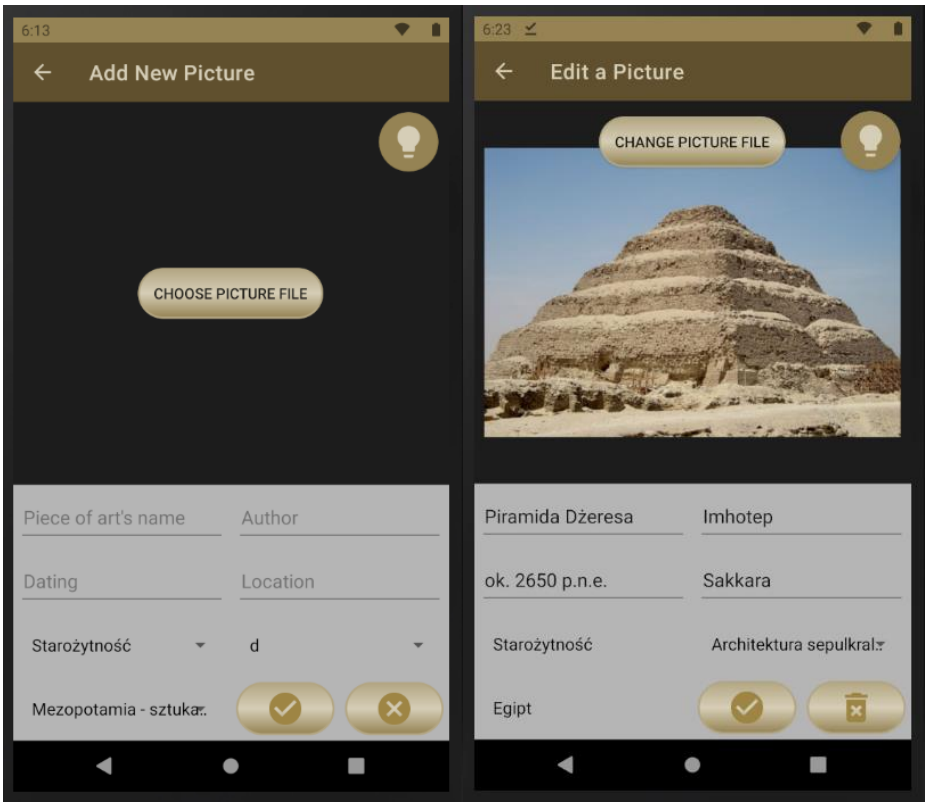
- udostępnia przycisk pozwalający na wyświetlenie oraz edycję Ciekawostki związanej z wybranym Obrazkiem i przekazuje treść ciekawostki Modelowi Fragmentu (rys. 24). Treść ciekawostki aktualizowana jest za pomocą Obserwatora powiązanego z obiektem *LiveData* zawierającym obiekt Ciekawostki.

- wyświetla plik graficzny przynależący do obrazka i pozwala na jego wybór oraz edycję (rys. 25), sczytuje ścieżkę do wybranego pliku graficznego i przekazuje ją Modułowi zarządzania Obrazkami.

Aktywność ta udostępnia również wspomnianą powyżej metodę „*refreshAddingPicture()*”, której wywołanie powoduje reset wszystkich pól aktywności (czyli widoków obrazu – *ImageView* oraz widoku przycisku Ciekawostki) oraz wywołanie nowego Fragmentu dodawania i edycji obrazka.



Rys. 24. Udostępniana przez Aktywność Pojedynczego obrazka możliwość edycji Ciekawostki przypisanej do obrazka, która od razu może być wyświetlona w ramach Modułu Uczenia Się dzięki zastosowaniu *LiveData* [opracowanie własne]



Rys. 25. Umożliwienie dodania nowego pliku graficznego, który zostanie przypisany do obiektu Obrazka (Aktywność Pojedynczego obrazka, w ramach Modułu Zarządzania Obrazkami). Możliwość edycji wyboru pliku graficznego [opracowanie własne]

3.7. Moduł Pobierania Zasobów

Moduł Pobierania Zasobów składa się z:

- jednej Aktywności (Activity),

- powiązanego z nią Modelu Widoku (ViewModel) oraz

- Odbiornika Emisji (BroadcastReceiver).

Moduł dostępny jest za pomocą:

- przycisku umieszczonego w Menu Głównym,

- przycisku wyświetlanego w ramach Modułu Uczenia się w przypadku, gdy baza danych nie zawiera żadnych Epok.

Wybranie dowolnego z wymienionych przycisków przenosi Użytkownika do aktywności zawierającej zestaw przycisków pozwalających na pobranie odpowiedniego pakietu danych do pobrania.

Gdy liczba udostępnianych pakietów wzrośnie, lista przycisków zastąpiona zostanie dedykowanym widokiem *RecyclerView* pobierającym aktualnie dostępną listę pakietów z serwera przypisanego aplikacji.

W obecnej wersji każdy z przycisków uruchamia link pozwalający na natychmiastowe i bezpośrednie pobranie pakietu danych skompresowanych za pomocą formatu ZIP. Żądanie pobrania pliku zostaje skierowane do systemowego Menedżera Pobierania, który następnie rozpoczyna proces pobierania wymaganego pliku. Jednocześnie wyświetlona zostaje wiadomość oraz ikona ładowania informujące Użytkownika o operacjach przeprowadzanych w tle (rys. 26).

Po pobraniu pliku System wysyła powiadomienie skierowane do Aplikacji, zawierające unikalny numer identyfikacyjny procesu pobierania dotyczącego zażądanego wcześniej pakietu. Zadaniem Odbiornika Emisji (*Broadcast Receiver*) jest odebranie tego powiadomienia i uruchomienie kodu mającego na celu rozpakowanie pobranego pakietu. Kod ten:

- powiadamia Użytkownika o tym, że plik został prawidłowo pobrany, lub że pobieranie zakończyło się niepowodzeniem (na przykład z powodu anulowania operacji z woli Użytkownika),

- rozpakowuje otrzymany pakiet typu ZIP, wyszukuje zawartą w nim listę Epok i Nurtów, dodaje wszystkie Epoki i Nurty, które znajdują się w pakiecie, lecz nie istnieją w bazie danych aplikacji (zostają porównane po nazwie),

- rozpakowuje katalog zawierający Obrazki oraz ich pliki graficzne. Pliki graficzne zapisane zostają w folderze dedykowanym danym aplikacji HaSZ umiejscowionym w pamięci urządzenia. Jeśli plik o tej nazwie już istnieje, nie jest on nadpisywany, ani nie jest tworzona jego kopia. Instancje obiektów typu Obrazek zostają następnie dodane do bazy danych aplikacji, przy czym jeśli metoda nie wykryje wymaganego Typu obrazka, również doda go do bazy danych,

- wywołuje metodę w Aktywności wyświetlającą powiadomienie o prawidłowo przeprowadzonym procesie rozpakowywania pakietu oraz ukrywającą ikonę ładowania przypisaną właśnie rozpakowanemu pakietowi.

Wywołanie metody w klasie Aktywności z poziomu klasy Odbiornika Emisji, której metoda nie jest zależna od aplikacji, lecz od komunikatu systemowego, jest możliwe dzięki stworzeniu i wykorzystaniu statycznej instancji Aktywności. Jednocześnie zmienne zawarte w klasie Aktywności deklarowane są jako statyczne (*static*), w przeciwieństwie do zmiennych zawartych w innych Aktywnościach (które zwykle przechowywane są w Modelach Widoku).

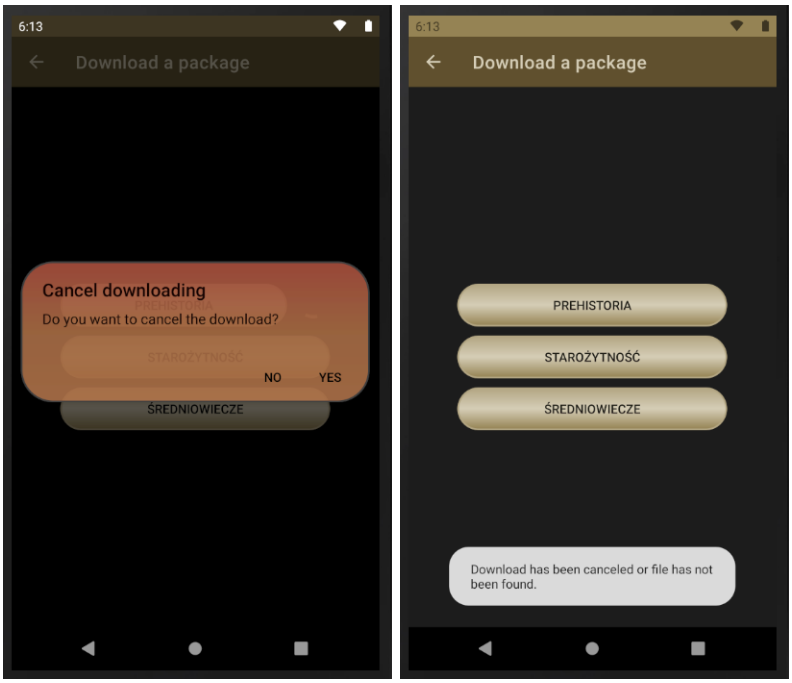
Rozwiązanie to było konieczne z powodu istnienia możliwości wyjścia z Aktywności, a następnie powrotu do niej. Model Widoku jest odporny na zmiany konfiguracyjne, takie jak zmiana położenia ekranu, jednak w momencie zamknięcia Aktywności wywołanego wciśnięciem przycisku powrotu, zostaje zniszczony razem z Aktywnością.

Tymczasem ważnym jest, aby indykatory pobierania pokazywały Użytkownikowi fakt przeprowadzania operacji w tle związanych z danym pakietem. Równie ważnym jest blokowanie przycisku pobierania w momencie, gdy paczka, do której przycisk prowadzi, jest właśnie pobierana – dzięki temu nie zachodzi ryzyko przypadkowego pobierania wielu jednakowych paczek wynikającego z frustracji Użytkownika pozbawionego informacji dotyczących aktualnie przeprowadzanych operacji.

Zmienne statyczne umożliwiają wyświetlanie zawsze aktualnych widoków w Aktywności, nawet jeśli zarówno Aktywność, jak i przynależący do niej Model Widoku zostaną wielokrotnie zniszczone i stworzone na nowo.

Rys. 26. Aktywność Modułu Pobierania Zasobów: indykator pobierania oraz wiadomość typu *Toast* [opracowanie własne]

W trakcie pobierania wybranego zasobu przycisk mu przypisany zmienia swoje przeznaczenie. Ponowne wybranie go wywołuje okno pytające Użytkownika, czy chce on anulować aktualnie przeprowadzane pobieranie wybranego pakietu (rys. 27, pierwszy zrzut ekranu). Jeśli Użytkownik zatwierdzi operację, do systemowego Menedżera Pobierania wysłane zostanie żądanie anulowania pobierania. Gdy żądanie zostanie spełnione, Odbiornik Emisji zostanie o tym powiadomiony i wywoła metodę Aktywności wyświetlającą komunikat informujący Użytkownika o zaprzestaniu pobierania (rys. 27, drugi zrzut ekranu).



Rys. 27. Aktywność Modułu Pobierania Zasobów: okno potwierdzenia operacji anulowania pobierania paczki oraz wiadomość wyświetlona po prawidłowym przeprowadzeniu tejże operacji [opracowanie własne]

Po prawidłowym pobraniu i rozpakowaniu pakietów, zawarte w nich Obrazki są już dostępne dla Użytkowników, którzy mogą przejść do Modułu Uczenia się, aby poznać zawartość pakietów. Wszystkie Obrazki udostępniają możliwość edycji przez Użytkownika za pomocą Modułu Zarządzania Obrazkami. W przyszłości dodana zostanie możliwość zapisywania konkretnych plików w katalogu wybranym przez Użytkownika. Na ten moment Użytkownik ma pełny dostęp do paczki z danymi w formacie .zip, dostępna jest bowiem z katalogu „Pobrane" istniejącego na urządzeniu końcowym, którego zawartość podlega pełnej kontroli Użytkownika.

3.8. Moduł Uczenia Się

Moduł Uczenia Się dostępny jest poprzez wybranie opcji „Learn" dostępnej w formie przycisku w Menu Głównym aplikacji HaSZ.

Moduł składa się z następujących elementów:

- Aktywności pozwalającej na wybór pożądanego elementu periodyzacyjnego, przypisanego jej Modela Widoku oraz Adaptera dla zawartego w niej widoku *RecyclerView*,

- Aktywności Galerii wyświetlającej wszystkie Obrazki należące do wybranego elementu periodyzacyjnego, przypisanego jej Modelu Widoku oraz Adaptera dla zawartego w niej widoku *RecyclerView*,

- Fragmentu Uczenia się wraz z przypisanym Modelem Widoku, działającego w ramach Aktywności Pojedynczego obrazka.

Pierwsza z aktywności pozwala na wybranie interesującej Użytkownika Epoki w sztuce z listy dostępnych (rys. 28). Lista zrealizowana jest za pomocą widoku *RecyclerView*, dla którego zaimplementowany został dedykowany Adapter. Wybranie jednego z dostępnych elementów zawierających Epokę w sztuce może spowodować dwa zachowania:

- jeśli Epoka posiada Nurty, wywołana zostanie nowa instancja tej samej Aktywności, zawierająca Listę Nurtów przynależących do wybranej Epoki, przy czym na górze ekranu pojawi się ikona pozwalająca na wyświetlenie oraz edycję Ciekawostki dotyczącej tejże Epoki (rys. 29),

- jeśli Epoka nie posiada Nurtów, wywołana zostanie Aktywność galerii, w której umieszczone zostaną Obrazki przynależące do tej Epoki (lecz, oczywiście, nie przynależących do żadnego Nurtu).

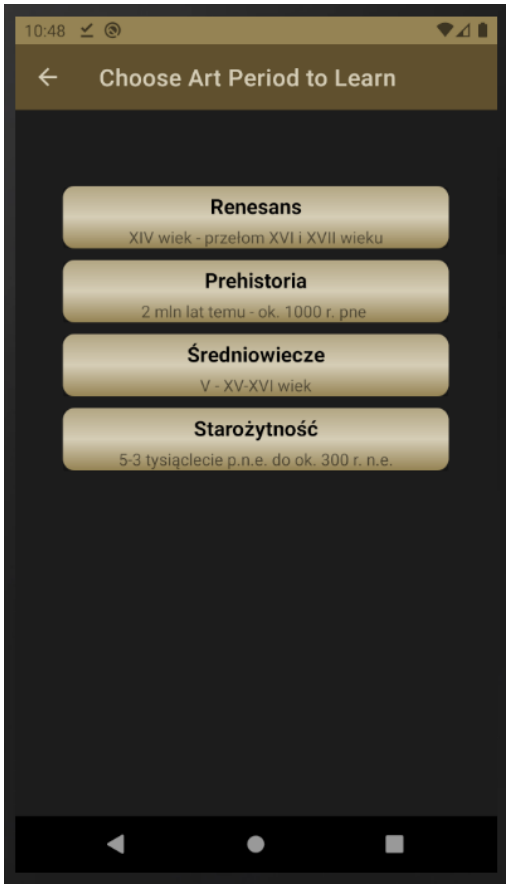
W przypadku, gdy Obserwator powiązany z obiektami *LiveData* zawierającymi listy elementów periodyzacyjnych i obrazków zwróci puste wartości, aktywności zareagują następująco:

- w przypadku braku Epok, Aktywność Wyboru wyświetli komunikat o braku Epok i zaproponuje dodanie nowej Epoki lub pobranie gotowej paczki z internetu (rys. 30, pierwszy zrzut ekranu),

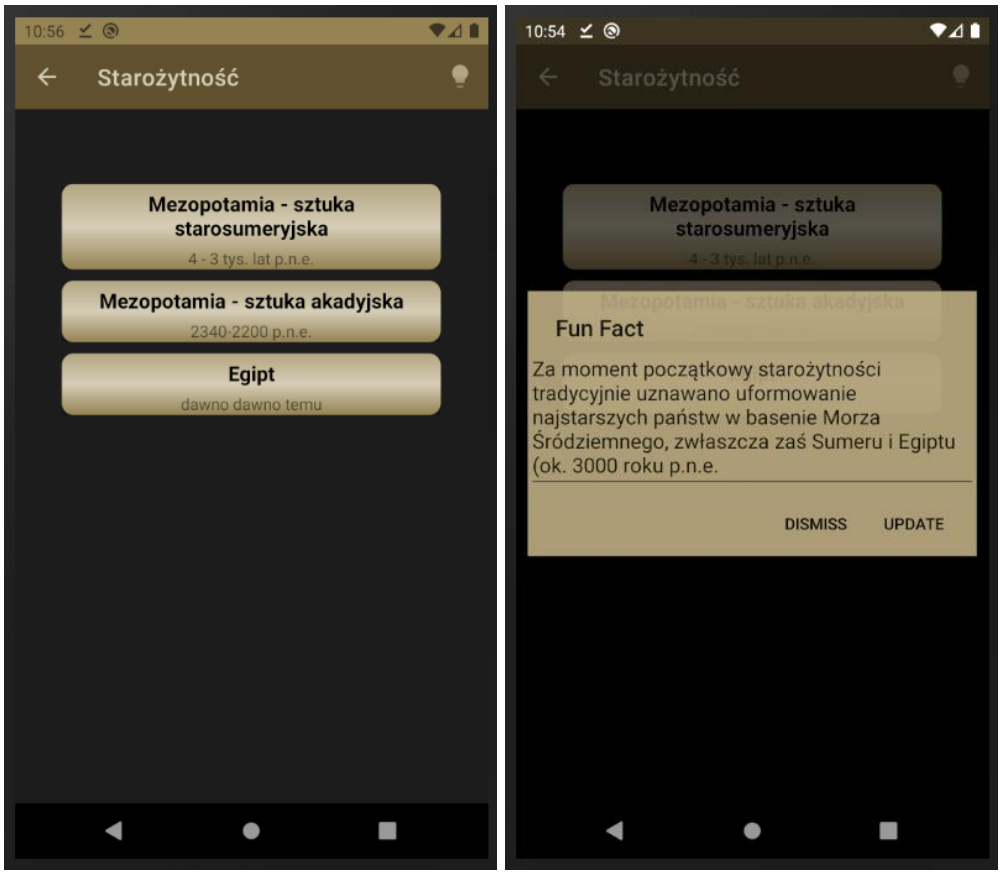
- w przypadku braku Nurtów oraz obrazków w ramach wybranej Epoki uruchomiona zostanie Aktywność Galerii, która zaproponuje Użytkownikowi dodanie nowego Nurtu lub nowego obrazka do wybranej Epoki (rys. 30, drugi zrzut ekranu), przy czym w intencji przesłanej wywoływanym Aktywnościom prześle informację o Epoce, z widoku której została wysłana prośba, aby aktywności dodawania nowego Nurtu oraz dodawania nowego obrazka mogły ustawić odpowiednią wartość na swoich widokach *Spinnerów*,

- w przypadku braku Obrazków w wybranym Nurcie uruchomiona zostanie Aktywność Galerii, która zaproponuje Użytkownikowi dodanie nowego obrazka

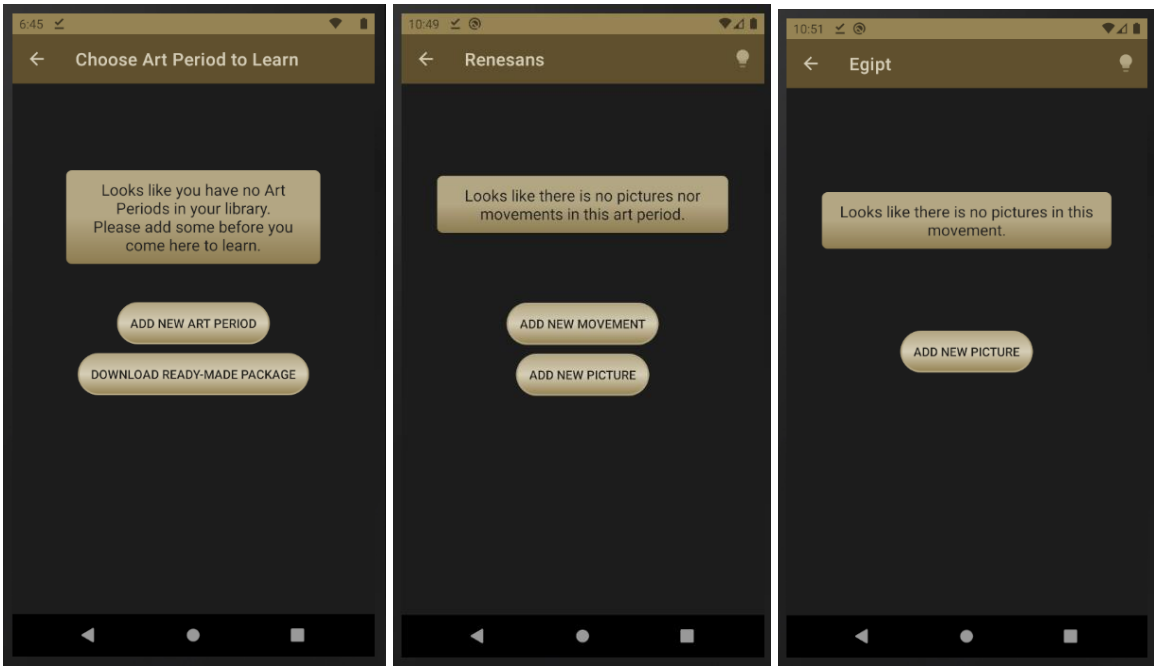
do wybranego Nurtu (rys. 30 , trzeci zrzut ekranu), przy czym w intencji przesłanej wywoływanej aktywności prześle informację o Nurcie, z widoku którego została wysłana prośba (a także Epoki, do której ów Nurt przynależy), aby Aktywność dodawania nowego obrazka mogła ustawić odpowiednią wartość na dwóch z trzech widoków *Spinnerów* (*Spinner* wyboru Epoki oraz *Spinner* wyboru Nurtu).



Rys. 28. Okno wyboru Epoki do nauki, Moduł Uczenia Się [opracowanie własne]



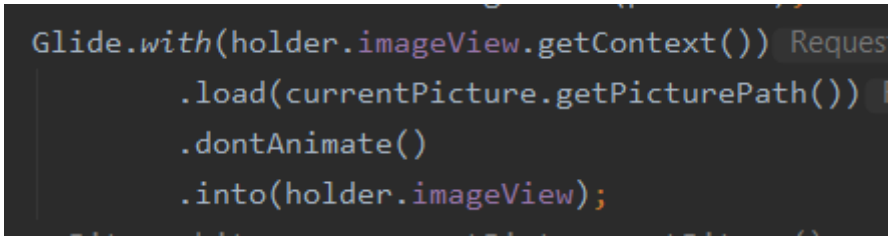
Rys. 29. okno wyświetlające zawartość wybranej Epoki, pozwalające na wyświetlenie i edycję Ciekawostki [opracowanie własne]



Rys. 30. Brak elementów w bazie danych. Propozycje dalszych działań [opracowanie własne]

Istnienie Obrazków w Epoce lub Nurcie spowoduje wyświetlenie widoku ich galerii w ramach Aktywności Galerii. Do wyświetlenia Obrazków używa ona widoku *RecyclerView* podzielonego na 4 kolumny.

Uzupełnianie *RecyclerView* w widoki poszczególnych Obrazków jest rolą przypisanego mu Adaptera, który z kolei używa biblioteki Glide (rys. 31) służącej do wczytywania plików graficznych na podstawie podanej ścieżki katalogowej pliku. Biblioteka *RoundedImageView* odpowiedzialna jest z kolei za wygładzanie rogów każdego z wyświetlanych obrazków oraz zaopatrzania każdego z ich widoków w cienką złotą ramkę.



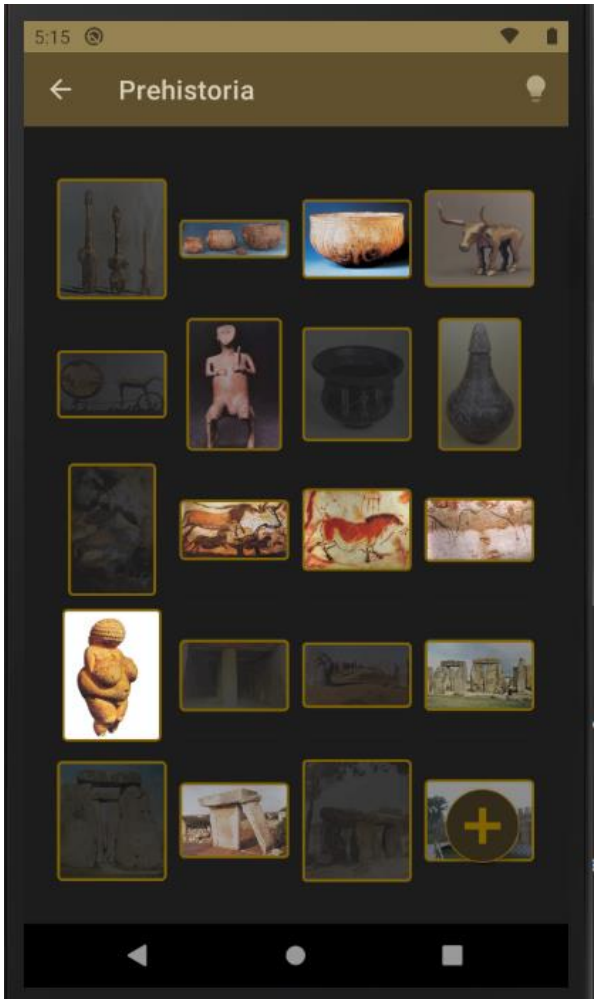
Rys. 31. Wykorzystanie biblioteki Glide w Aktywności Galerii [opracowanie własne]

Adapter Galerii posiada nietypową funkcję charakterystyczną dla aplikacji HaSZ, mianowicie oblicza widoczność poszczególnych obrazów na podstawie stopnia znajomości obrazu prezentowanego przez Użytkownika (rys. 32).

Każdy obiekt Obrazka posiada przypisany własny stopień znajomości, który wynosi od 0 do 10. Modyfikuje się go za pomocą wybierania przycisków „Umiem” oraz „Nie umiem” w Module Uczenia Się, w Aktywności Pojedynczego obrazka wyposażonej we Fragment służący do nauki. Każdy Obrazek podczas tworzenia przypisany ma poziom znajomości równy 0. Użytkownik nie może modyfikować stopnia znajomości obrazka w sposób jawny, nie ma też żadnego bezpośredniego dostępu do tego pola w bazie danych.

Stopień znajomości każdego z obrazków służy do obliczenia widoczności widoków zawierających prezentujące je pliki graficzne w Aktywności Galerii. Im mniejszy stopień znajomości obrazka, tym mniej widoczny jest widok zawierający jego graficzną reprezentację. W związku z tym nauka w aplikacji HaSZ odbywa się poprzez „odkrywanie” poszczególnych Obrazków i tym samym czynienie ich lepiej widocznymi.

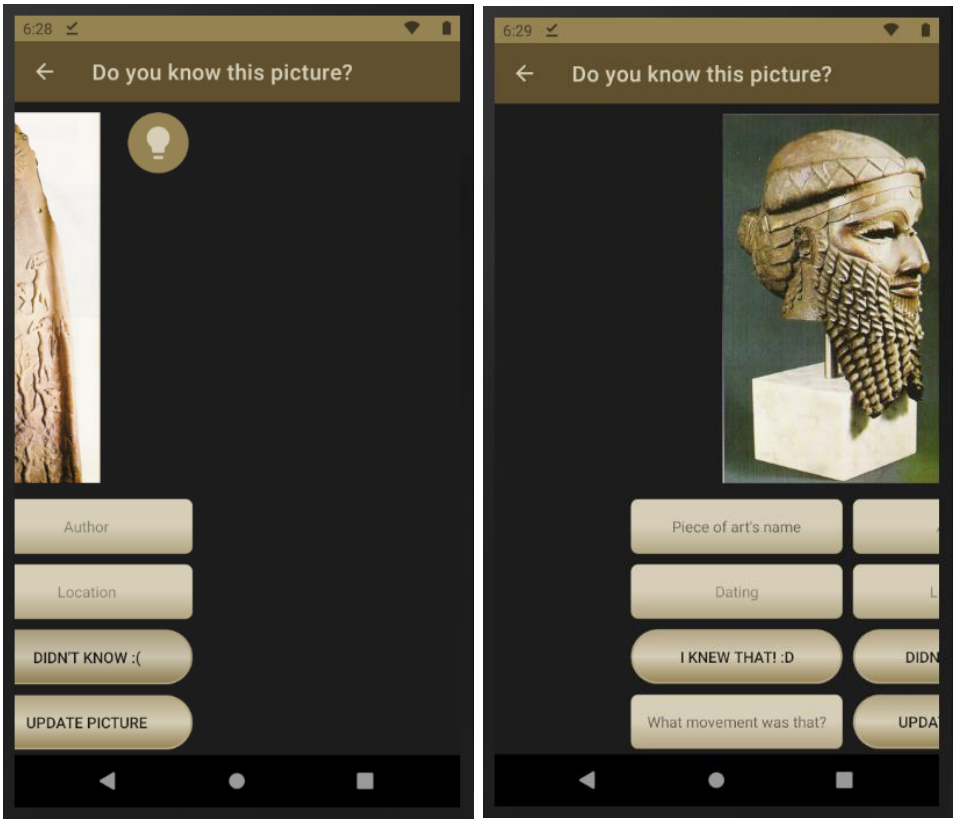
W przyszłości zaimplementowana zostanie funkcja aplikacji degradująca znajomość poszczególnych Obrazków na podstawie czasu, który upłynął od ostatniej interakcji z obrazkiem. W założeniu ma to zachęcić Użytkowników do regularnego zaglądania do obrazków.



Rys. 32. widok galerii Epoki Prehistoria. Widoczność poszczególnych Obrazków uzależniona jest od stopnia znajomości poszczególnych obiektów prezentowanego przez Użytkownika a zapisanego w bazie danych aplikacji [opracowanie własne]

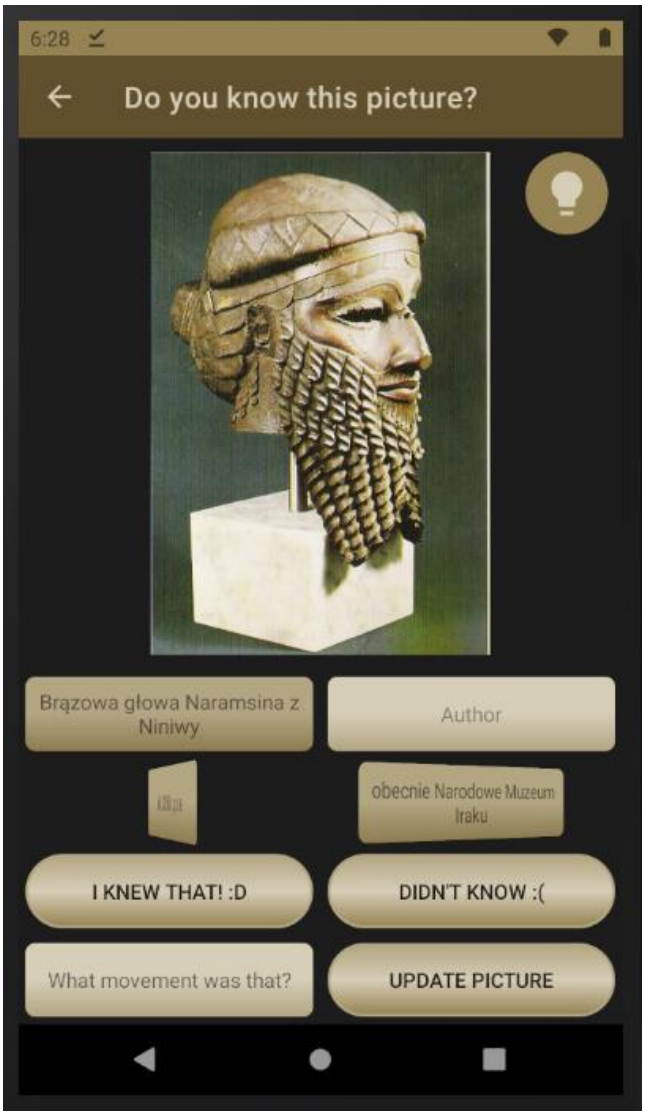
Wybranie jednego z obrazów znajdujących się w widoku galerii pozwala na wyświetlenie go w powiększeniu w ramach Aktywności Pojedynczego obrazka. Wysyłana przez Aktywność Galerii intencja zawiera w sobie informację dotyczące celu uruchomienia Aktywności, przekazuje również listę numerów identyfikacyjnych wszystkich elementów zawartych w galerii. Dzięki temu Aktywność Pojedynczego obrazka może pozwolić Użytkownikami na nawigację poprzez wszystkie Obrazki zawarte w galerii Epoki / Nurtu

bez potrzeby wyłączania Aktywności. Użytkownik może nawigować pomiędzy Obrazkami poprzez przesuwanie ich w lewo lub w prawo (rys. 33). Również wybranie przez niego przycisków „Umiem” oraz „Nie umiem” spowoduje wczytanie kolejnego elementu z dostarczonej listy.



Rys. 33. Nawigacja w ramach listy Obrazków należących do aktualnie przeglądanej galerii. Moduł Uczenia Się [opracowanie własne]

Wczytywany Fragment prezentuje Użytkownikowi karty zawierające wszystkie dane o obrazku, natomiast ikona znajdująca się w prawym górnym rogu Aktywności pozwala na podgląd jego Ciekawostki (rys. 35). Karty zawarte we Fragmencie można odwracać poprzez kliknięcie w nie – powoduje to wywołanie animacji odwracania karty i ujawnia wybraną informację na temat Obrazka (rys. 34). Dzięki temu Użytkownik może spróbować przypomnieć sobie informacje, zanim je zobaczy, co pozwala mu później zdecydować o poziomie swojej wiedzy na temat wybranego obrazka.



Rys. 34. podgląd Obrazka w ramach Modułu Uczenia Się, animacja odwracania poszczególnych kart z informacjami o obrazku [opracowanie własne]

Jak wspomniano podczas omawiania Modułu Zarządzania Obrazkami, podgląd Obrazka w ramach Modułu Uczenia Się pozwala na przejście do edycji wybranego obrazka. Oznacza to wymianę jedynie Fragmentu w przeznaczonym do tego polu Aktywności Pojedynczego obrazka, nie wymaga otwierania nowej aktywności i wczytywania wszystkich informacji na nowo. Aktywność nie musi przeładowywać na nowo pliku graficznego w celu przejścia do okna edycji obrazka, informacje na jego temat są natomiast przekazywane przez nią nowej instancji Fragmentu edycji obrazka poprzez plik tobołka *Bundle*.

Rys. 35. Ciekawostka dotycząca aktualnie przeglądanego obrazka widoczna w ramach Modułu Uczenia Się [opracowanie własne]

3.9. Moduł Testowy

Dostępny jest z Menu Głównego, przejść do niego można poprzez wybranie przypisanej mu opcji.

Opiera się na następujących elementach:

- Aktywność wyświetlająca rozwijaną listę wszystkich Epok i przynależących do nich Nurtów,

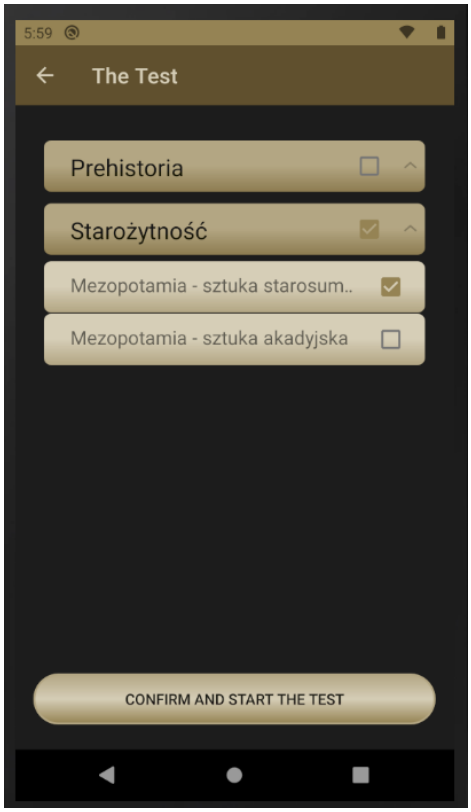
- klasa Adaptera listy wspomaganą przez 2 klasy opakowującej elementy periodyzacji na potrzeby Adaptera,

- klasa Widoku Modelu przypisana do aktywności,

- Fragment Testowy działający w ramach Aktywności Pojedynczego obrazka.

Zadaniem Aktywności Listy jest zebranie wymagań Użytkownika dotyczących wyboru elementów periodyzacyjnych, o których zawartości Użytkownik chce sprawdzić poziom swojej wiedzy. Listę tę uzupełnia klasa Adaptera zawierająca zestaw metod kreujących obiekty rodzica z obiektów Epok oraz obiekty dzieci z obiektów Nurtów. Widok rozwijanej listy wyświetla najpierw wszystkie elementy rodziców, czyli zawierające nazwy Epok, a po kliknięciu w wybrany element rozwija mniejszą listę w ramach listy głównej, zawierającą dzieci wybranego elementu, czyli Nurty przynależące do wybranej Epoki.

Każdy z elementów posiada własne pole wyboru, które Użytkownik może zaznaczyć, jeśli chce, aby zawartość elementu znalazła się w teście (rys. 36). Zaznaczenie elementu Epoki oznacza automatyczne zaznaczenie wszystkich zawartych w niej Nurtów, odznaczenie natomiast prowadzi do odznaczenia tychże Nurtów. Natomiast zaznaczenie Nurtu oznacza automatyczne zaznaczenie pola wyboru Epoki, do której Nurt przynależy, nie ciągnie jednak to za sobą zaznaczenia reszty Nurtów do Epoki należących.



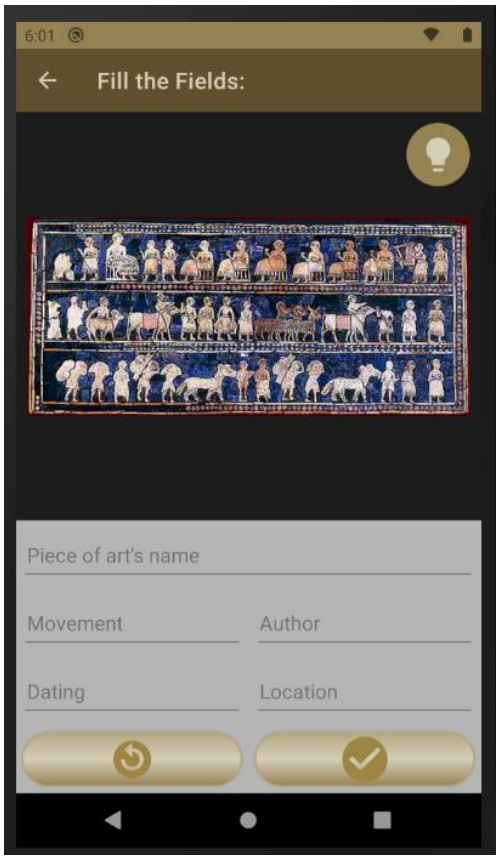
Rys. 36. Wybór elementów periodyzacyjnych do testu, Moduł Testowy [opracowanie własne]

Wciśnięcie przycisku znajdującego się na dole ekranu oznacza zatwierdzenie wyboru Epok i Nurtów i prowadzi do wygenerowania listy Obrazków, z której przeprowadzony zostanie test (jeśli lista będzie pusta, wyświetlony zostanie stosowny komunikat i okno nie ulegnie zmianie). Lista pozyskiwana jest przy pomocy dedykowanej metody zawartej w obiekcie PictureDAO:

```
@WorkerThread
@Query("SELECT * FROM Picture WHERE pictureArtPeriod IN (:artPeriodIds) OR pictureMovement IN (:pictureMovementIds)")
List<Picture> getPictureListByArtPeriodIdOrMovementIdSync(List<Integer> artPeriodIds, List<Integer> pictureMovementIds);
```

Rys. 37. Zapytanie zwracające listę numerów identyfikacyjnych wszystkich Obrazków należących do wybranych Epok i Nurtów w sztuce [opracowanie własne]

Odnośnik do tej metody zawarty jest w Repozytorium Obrazków, którego instancję posiada Model Widoku Modułu Testowego. Metoda nie wykorzystuje obiektów *LiveData*. Uruchamiana jest natomiast w wątku w tle. W trakcie wykonywania niezbędnych obliczeń wątek główny nie jest blokowany. Oczekuje na wynik obliczeń w postaci listy numerów ID, a gdy zostanie ona zapewniona, tasuje ją. Następnie uruchamiana jest Aktywność Pojedynczego obrazka, która na podstawie informacji przekazanych w Intencji wywołuje Fragment Testowy i wyświetla plik graficzny pierwszego obrazka z listy (rys. 38).



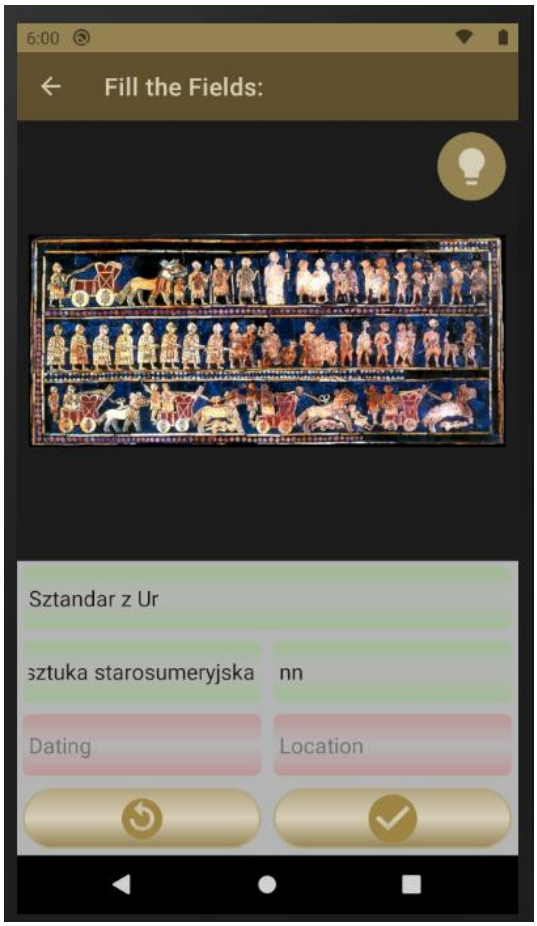
Rys. 38. Aktywność Pojedynczego obrazka z wczytanym Fragmentem Testowym, Moduł Testowy [opracowanie własne]

Aktywność Pojedynczego obrazka jak zwykle wyświetla plik graficzny przypisany do aktualnie prezentowanego obrazka, pozwala też na podgląd Ciekawostki dotyczącej tegoż Obrazka. Fragment Testowy natomiast pozwala na wpisywanie informacji dotyczących Obrazka w przygotowane ku temu pola, podobnie jak Fragment Dodawania i Edycji obrazka. Różnice obejmują brak *Spinnerów* – Nurt należy wpisać samodzielnie – oraz dwa przyciski, z których jeden pozwala na usunięcie dotychczas wpisanych danych we wszystkich polach oraz przycisk zatwierdzający odpowiedzi.

Wybranie przycisku zatwierdzającego sprawi, że pól nie będzie już można edytować zostaną one natomiast podświetlone (rys. 39):

- na kolor czerwony, jeśli udzielona odpowiedź nie jest zgodna z informacją zawartą w bazie danych,

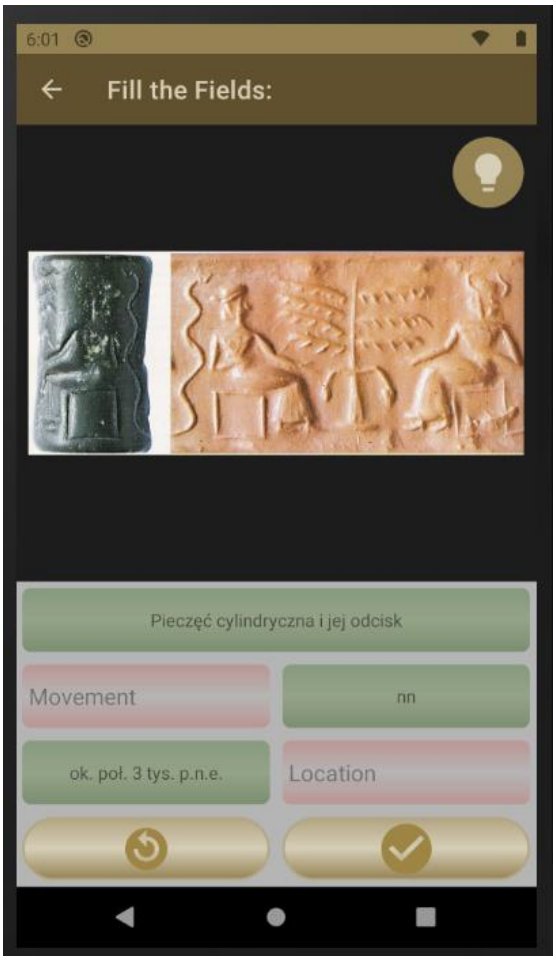
- na kolor zielony, jeśli informacje są w pełni zgodne.



Rys. 39. Podświetlenie udzielonych wypowiedzi, Moduł Testowy [opracowanie własne]

W tym momencie Użytkownik może sprawdzić prawidłowe odpowiedzi poprzez kliknięcie w kartę informacji, podobnie jak robił to podczas nauki. Karta odwróci się, ukazując prawidłową odpowiedź (rys. 40). Dzięki temu mechanizmowi Użytkownik sam może zdecydować, czy udzielona przez niego odpowiedź była wystarczająco blisko prawidłowej i czy czuje się na siłach podjąć prawdziwy test w ramach nauki w liceum bądź na studiach.

Aplikacja porównuje łańcuchy znaków dosłownie, co oznacza, że różnica rzędu jednej kropki oznaczać będzie nieprawidłową odpowiedź i brak punktu za Obrazek. W związku z tym wyniki testu nie są nigdzie zapisywane – nie są bowiem wystarczająco miarodajne. W przyszłości zostanie zaimplementowany w tym Module mechanizm służący porównywaniu łańcuchów znaków, przez co wynik wyświetlany będzie w formie procentów. Na tę chwilę to do Użytkownika należy ostateczna decyzja dotycząca pewności siebie związanej ze znajomością danego obrazka.

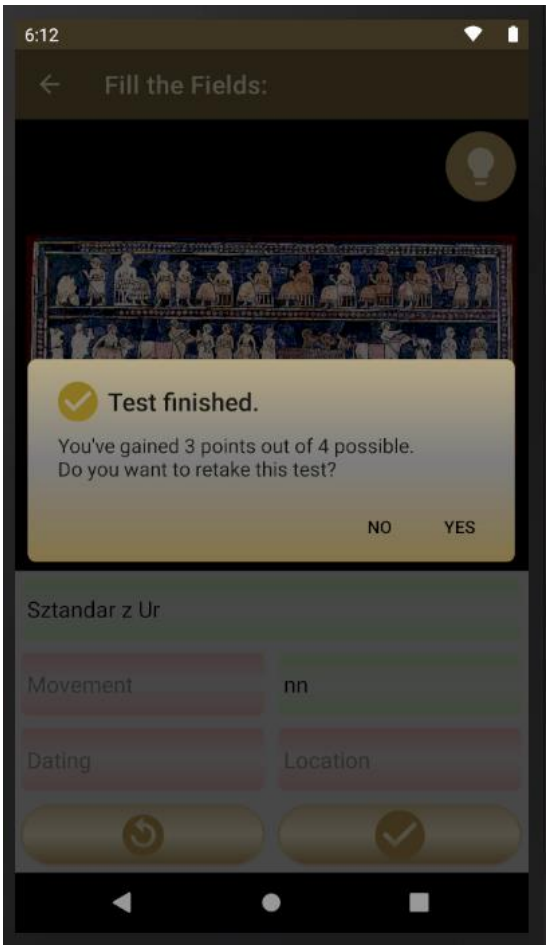


Rys. 40. Sprawdzanie prawidłowych odpowiedzi – ciemnozielone pola odsłaniają się po odwróceniu karty i zawierają prawidłowe odpowiedzi [Opracowanie własne].

Za każdy z obrazków może zostać przyznany jeden punkt. Punkty przyznawane są jedynie za Obrazki, których wszystkie informacje zostały wypełnione w pełni prawidłowo. Suma punktów przechowywana jest w Modelu Widoku Aktywności Pojedynczego Obrazka i aktualizowana z każdym kolejnym Obrazkiem biorącym udział w teście.

Ponowne wciśnięcie przycisku zatwierdzającego przenosi Użytkownika do widoku kolejnego Obrazka z listy testowej. Brak jest możliwości powrotu do poprzednich Obrazków. Są one usuwane z listy testowej.

Na koniec testu Użytkownikowi prezentowany jest wynik w formie komunikatu (rys. 41), ukazany w formacie „ilość zdobytych punktów / ilość punktów możliwych do zdobycia”, przy czym pierwsze oznacza liczbę Obrazków, których opisy zostały całkowicie poprawnie uzupełnione, drugie natomiast jest ilością wszystkich Obrazków występujących w danym teście.



Rys. 41. Okno wyniku testu, Moduł Testowy [opracowanie własne]

Komunikat informujący Użytkownika o zakończeniu testu pozwala na powtórzenie go od początku z tą samą listą Obrazków. Lista przechowywana jest w Modelu Widoku w formie Listy Zapasowej. Jeśli Użytkownik zdecyduje się na powtórzenie testu, Lista Zapasowa ulega przetasowaniu, a jej zawartość kopiowana jest do Listy Testowej, a wynik jest zerowany. Następnie wyświetlany jest pierwszy Obrazek z listy i test rozpoczyna się ponownie.

Test można powtarzać dowolną ilość razy. Jego wynik nie jest w aktualnej wersji wystarczająco miarodajny, aby na jego podstawie obniżać stopień znajomości poszczególnych Obrazków, a zatem degradować poziom ich widoczności w widoku Galerii. W przyszłości wraz z implementacją bardziej wydajnych algorytmów porównujących będzie to możliwe i zostanie zaimplementowane.

3.10. Katalog pomocniczy

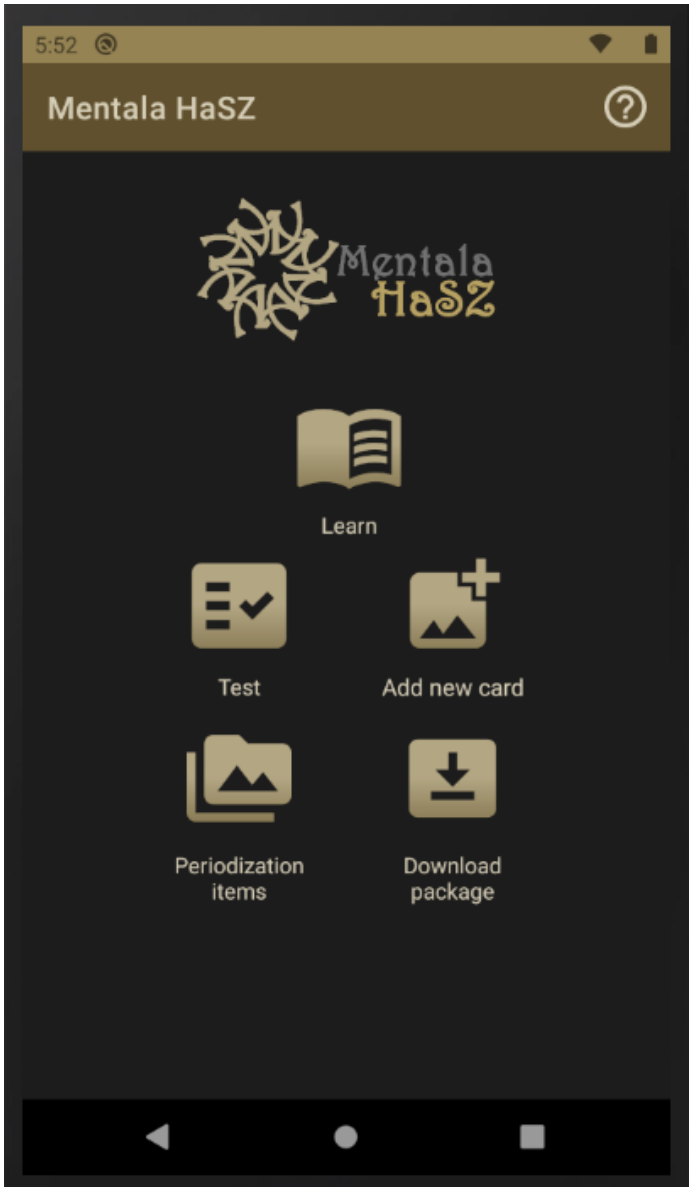
Katalog „Utils” zawiera serię klas pomocniczych wykorzystywanych przez pozostałe części aplikacji HaSZ w celu zachowania spójności i jednolitości (tu zawarte są statyczne wartości stałe (static final) wykorzystywane na przestrzeni całego projektu) oraz ograniczenia powtarzania tego samego kodu w różnych klasach (takie jak metody służące wywoływaniu nowych Aktywności wraz z przekazywaniem im szeregu zmiennych poprzez intencje).

Wykorzystanie klas pomocniczych nie tylko zmniejsza ilość zbędnych powtórzeń w kodzie, ale przede wszystkim nie pozwala na błędy polegające na pominięciu przekazania wymaganej wartości w jednym z wielu miejsc, w których jest ona przekazywana. Szczególnie często błędy tego typu występują w przypadku wprowadzania zmian w projekcie – jeśli nie istnieje jedna wspólna metoda wymagająca konkretnego zestawu wartości, bardzo łatwo jest przeoczyć co najmniej jedno miejsce, w którym wywoływanie należy o ową wartość uzupełnić.

3.11. Menu główne

Menu Główne aplikacji HaSZ wyświetla się bezpośrednio po uruchomieniu aplikacji (rys. 42).

Zawiera logo aplikacji, przycisk pozwalający na uzyskanie informacji o aplikacji oraz zestaw pięciu przycisków nawigacyjnych, dzięki którym Użytkownik może uruchomić każdy z opisanych z poprzednich podrozdziałach Modułów.



Rys. 42. Główne Menu aplikacji HaSZ [opracowanie własne]

3.12. Permisje

Aby prawidłowo działać, aplikacja HaSZ wymaga dwóch pozwoleń (*permissions*):

- pozwolenia dostępu do pamięci urządzenia w celu odczytania plików graficznych Obrazków,

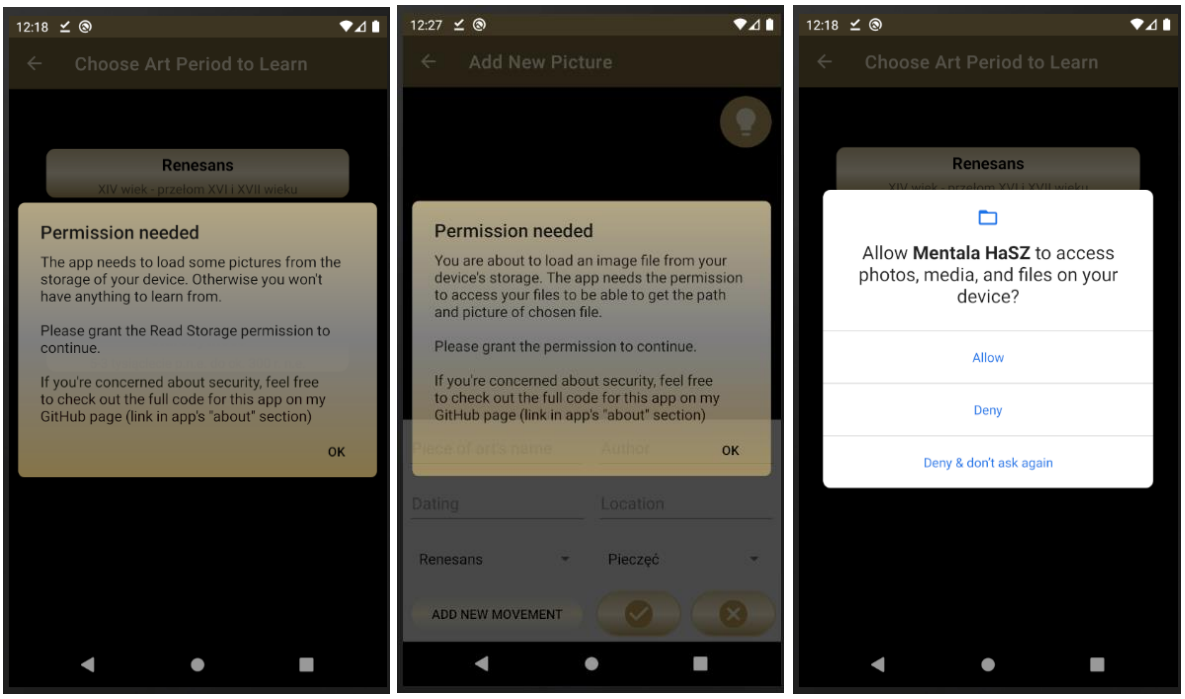
- pozwolenia na korzystanie z Internetu w celu pobrania paczek z obrazkami.

O ile druga permisja jest wymagana tylko w przypadku korzystania z Modułu Pobierania Zasobów, o tyle pierwsza z permisji jest niezbędna do prawidłowego działania aplikacji.

Aplikacja HaSZ nie kopiuje plików graficznych do wewnętrznego folderu aplikacji, gdyż oznaczałoby to znaczne marnowanie pamięci urządzenia, a tymczasem urządzenia mobilne zwykle nie są wyposażone w naprawdę duże zasoby pamięciowe. Zatem zarówno wybór plików graficznych, które zostaną przypisane poszczególnym Obrazkom w trakcie ich dodawania do bazy danych, jak i późniejszy odczyt w trakcie nauki lub sprawdzania swojej wiedzy, wymaga dostępu do pamięci urządzenia, z którego aplikacja pobiera plik graficzny, który następnie wyświetla na ekranie urządzenia przy pomocy biblioteki Glide.

Aplikacja HaSZ sprawdza dostęp do permisji za każdym razem, gdy potrzebuje ich do kontynuowania działania. W przypadku braku potrzebnego pozwolenia wyświetla komunikat z uzasadnieniem prośby uzyskania konkretnego dostępu (rys. 43, zrzuty ekranu 1 i 2), a następnie wywołuje systemowe okno pozwalające na decyzję, czy pozwolenie zostanie udzielone, czy też nie (rys. 43, zrzut ekranu 3).

W przypadku braku wymaganych pozwoleń aplikacja HaSZ przechodzi do procesu zwanego wdzięczną degradacją (*graceful degradation*), co oznacza, że aplikacja nadal działa, jednak część jej funkcjonalności nie jest dostępna. W przypadku aplikacji HaSZ jedynym modułem działającym w pełni bez dodatkowych pozwoleń jest Moduł Zarządzania Elementami Periodyzacji, on też pozostaje aktywny w przypadku braku udzielenia permisji.



Rys. 43. przykłady uzasadnień dotyczących dodatkowych pozwoleń wykorzystywane w aplikacji HaSZ [opracowanie własne]

3.13. Kolorystyka projektu

Zgodnie z założeniem projektu, kolorystyka aplikacji HaSZ utrzymana została w stonowanych, przyciemnionych barwach (rys. 44). Aplikacja została pozbawiona wszelkich zbędnych elementów, nie posiada wyrazistych, ani kontrastujących ze sobą kolorów. Wszelkie informacje wyświetlone są z wykorzystaniem szablonów o łagodnym doborze kolorów.

W przeciwieństwie do okien przynależących do Systemu (rys. 43, trzeci zrzut ekranu), okna prezentowane przez aplikację HaSZ nigdy nie są oparte na białym tle, dzięki czemu nie dostarcza ona zbędnych bodźców świetlnych swoim Użytkownikom.

Rys. 44. zestawienie kolorów użytych w aplikacji HaSZ [opracowanie własne]

PODSUMOWANIE

Aplikacja HaSZ spełnia wszystkie sporządzone dla niej wymagania funkcjonalne oraz niefunkcjonalne. Jest aplikacją prostą, a jednocześnie w pełni funkcjonalną. Nie wysyła nadmiaru bodźców swoim Użytkownikom, nie prezentuje im też nadmiaru informacji, dzięki czemu walczy ze zjawiskiem przeciążenia informacyjnego wśród swoich Użytkowników. Pozwala w prosty i przyjemny sposób uczyć się historii sztuki zarówno uczniom i studentom, jak i nieprofesjonalnym pasjonatom. W przyszłości zostanie uzupełniona o funkcjonalności wspierające jej działanie, wśród których nie będzie jednak żadnej zbędnej lub niepowiązanej bezpośrednio z głównym zadaniem aplikacji. Istniejące funkcjonalności zostaną natomiast ulepszone, dzięki czemu aplikacja będzie jeszcze lepiej spełniała swoje zadanie.

Perspektywy rozwojowe

Zaprezentowany projekt jest pierwszą w pełni działającą wersją aplikacji HaSZ, lecz z pewnością nie ostatnią. Istnieje wiele usprawnień i udogodnień, które mogą zostać zaimplementowane w aplikacji, podobnie jak kilka funkcjonalności wykazujących potencjał wsparcia jej działania bez przeciążenia programu zbędnymi dodatkami. Oznacza to dodanie następujących usprawnień i funkcjonalności:

- wsparcie wielu języków,
- pełne wsparcie starszych systemów (podstawowe funkcje są już zaimplementowane w sposób, który powinien im pozwalać na działanie na starszych wersjach systemu Android, jednak nie przeprowadzono jeszcze żadnych testów i związanych z nimi poprawek w tym zakresie),
- rozszerzenia dotyczące interfejsu użytkownika: wsparcie horyzontalnego układu ekranu, wprowadzenie jasnego motywu, wprowadzenie możliwości wyboru koloru przodującego w palecie kolorystycznej aplikacji
- sortowanie epok zgodnie z czasem oraz wprowadzenie możliwości swobodnego układania ich przez Użytkownika za pomocą przeciągania Epok w aktywności wyświetlającej ich listę,

- funkcja degradująca znajomość poszczególnych Obrazków na podstawie czasu, który upłynął od ostatniej interakcji z obrazkiem,

- rozbudowa Modułu Testowego o mini-gry oraz quiz, implementacja funkcji procentowo porównujących łańcuchy znaków, dzięki czemu uzyskanie punktu za Obrazek podczas testu będzie bardziej realne,

- dodanie możliwości tworzenia kopii zapasowej bazy danych oraz własnych paczek, którymi następnie będzie można podzielić się z przyjaciółmi i kolegami,

- wyszukiwanie Obrazków po nazwie / autorze / elementach periodyzacyjnych (na przykład wyświetlenie zestawienia wszystkich Obrazków występujących we wszystkich Nurtach wybranej Epoki),

- sortowanie Obrazków po Typie / autorze.

Wszystkie wymienione funkcjonalności nie są niezbędne do prawidłowego działania aplikacji, jednak posiadają predyspozycje do znacznej poprawy jej przystępności względem Użytkowników.

Wnioski

Realizacja projektu i implementacji aplikacji HaSZ okazała się zarówno dobrą zabawą, jak i poważnym wyzwaniem. Zastosowane rozwiązania często powstawały przez długi czas i dopiero po wielu nieudanych próbach były w stanie uzyskać swój finalny, dobrze działający kształt.

Autorka nauczyła się wiele podczas realizacji projektu, szczególnie w temacie działania systemu Android oraz znacząco rozwinęła swoje umiejętności posługiwania się językiem Java oraz programowania w ogóle. Wybór niniejszej aplikacji jako projektu dyplomowego okazał się wspaniałym pomysłem, którego wykonanie wspomogło rozwój Autorki.

Jednocześnie aplikacja ma realną szansę na zdobycia popularności wśród dedykowanych Użytkowników, na co Autorka szczerze liczy.

LITERATURA

[1] D. G. i. D. Griffiths, Rusz głową! Android - programowanie aplikacji, Gliwice: HELION, 2016.

[2] S. Molitorys, „justgeek.it," JustJoin.it, [Online]. Available: https://geek.justjoin.it/ide-jak-wykorzystac-to-narzedzie-do-codziennej-pracy. [Data uzyskania dostępu: 13 09 2021].

[3] B. M. K. W. Stanisław Wrycza, Język UML 2.0 w modelowaniu systemów informatycznych, Gliwice: Helion, 2006.

[4] „Java Lambda Expressions," W3Schools, [Online]. Available: https://www.w3schools.com/java/java_lambda.asp. [Data uzyskania dostępu: 02 09 2021].

[5] A. Code, „Wzorce Architektoniczne - MVVM," [Online]. Available: http://androidcode.pl/blog/wzorce/mvvm/. [Data uzyskania dostępu: 13 09 2021].

[6] „Wikipedia. Obserwator (wzorzec projektowy)," [Online]. Available: https://pl.wikipedia.org/wiki/Obserwator_(wzorzec_projektowy). [Data uzyskania dostępu: 13 09 2021].

[7] [Online]. Available: https://myenv.net/blog/android-architecture-components-livedata/. [Data uzyskania dostępu: 13 08 2021].

[8] „Dokumentacja komponentów systemu Android. LiveData.," [Online]. Available: https://developer.android.com/topic/libraries/architecture/livedata. [Data uzyskania dostępu: 20 08 2021].

[9] „Android.com.pl Cykl życia aplikacji," [Online]. Available: https://android.com.pl/programowanie/152192-cykl-zycia-aplikacji/. [Data uzyskania dostępu: 10 09 2021].

[10] „Wikipedia - Wyciek pamięci," [Online]. Available: https://pl.wikipedia.org/wiki/Wyciek_pami%C4%99ci. [Data uzyskania dostępu: 31 08 2021].

[11] „Wikipedia - Data Access Object," [Online]. Available: https://pl.wikipedia.org/wiki/Data_Access_Object. [Data uzyskania dostępu: 13 07 2021].

[12] „Wikipedia - Hermetyzacja (informatyka)," [Online]. Available: https://pl.wikipedia.org/wiki/Hermetyzacja_(informatyka). [Data uzyskania dostępu: 26 08 2021].

[13] „Dropbox,” [Online]. Available: https://www.dropbox.com/pl/business/resources/what-is-a-zip-file. [Data uzyskania dostępu: 01 09 2021].

[14] „Android4Devs - wstęp do intencji,” [Online]. Available: http://www.android4devs.pl/2011/07/wstep-do-intencji-intents/. [Data uzyskania dostępu: 25 07 2021].

[15] „ARPU Brothers - Jak zarabiać na aplikacjach mobilnych?,” [Online]. Available: https://arpubrothers.pl/blog/zarabianie-na-aplikacjach-mobilnych/. [Data uzyskania dostępu: 16 07 2021].

[16] A. M. Davis, „Publisher’s Paradox: Information Overload,” [Online]. Available: https://www.slideshare.net/tpldrew/publishers-paradox-information-overload. [Data uzyskania dostępu: 13 09 2021].

[17] C. E. Wilson, „Information discrimination: a human habit.,” *Canadian Journal of Information Science,* tom 1, p. 59–64, 1976.

[18] G. W. Amanda Hall, „Information overload within the health care system: a literature review,” *Health Information and Libraries Journal,* tom 21, p. 102–108, 2004.

[19] M. A. F. S. N. P. V. P. S. M. P. K. S. D. a. K. R. Ashutosh Kumar, „Addictive Influences and Stress Propensity in Heavy Internet Users: A Proposition for Information Overload Mediated Neuropsychiatric Dysfunction,” *Current Psychiatry Reviews,* tom 13, 2017.

[20] N. G. D. &. S. S. Dadashi, „Seeing the woods for the trees: the problem of information inefficiency and information overload on operator performance,” *Cogn Tech Work,* tom 19, p. 561–570, 2017.

[21] „Strona Artly na witrynie Google Play,” [Online]. Available: https://play.google.com/store/apps/details?id=com.pavelkozemirov.guesstheartist&hl=pl&gl=US. [Data uzyskania dostępu: 01 09 2021].

[22] „Aplikacja History of Art w sklepie Google Play,” [Online]. Available: https://play.google.com/store/apps/details?id=com.historyofart. [Data uzyskania dostępu: 01 06 2021].

[23] „Aplikacja Google Arts & Culture w sklepie Google Play,” [Online]. Available: https://play.google.com/store/apps/details?id=com.google.android.apps.cultural. [Data uzyskania dostępu: 01 06 2021].

[24] „Aplikacja DailyArt w sklepie Google Play,” [Online]. Available: https://play.google.com/store/apps/details?id=com.moiseum.dailyart2. [Data uzyskania dostępu: 01 06 2021].

[25] „Aplikacja Artier w sklepie Google Play,” [Online]. Available: https://play.google.com/store/apps/details?id=com.vipulasri.artier. [Data uzyskania dostępu: 01 06 2021].

[26] „Inżynieria oprogramowania,” [Online]. Available: https://fizyka.ujk.edu.pl/pl/files/lectures/Inzynieria_oprogramowania/UJK-IO-FazaOkrWym.pdf. [Data uzyskania dostępu: 01 09 2021].

[27] *The Institute of Electrical and Electronics Engineers Standards Board: Recommended Practice for Architectural Description of Software-Intensive Systems,* ISO/IEC 42010:2007(E) IEEE Std 1471-2000, 2007.

[28] „Geeks for Geeks. Top programming languages for Android app development,” [Online]. Available: https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/. [Data uzyskania dostępu: 08 09 2021].

[29] „TechCrunch - Kotlin is now Google's preferred language for Android app development,” [Online]. Available: https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/. [Data uzyskania dostępu: 05 09 2021].

[30] „Dokumentacja Android - Room persistence library,” [Online]. Available: https://developer.android.com/training/data-storage/room. [Data uzyskania dostępu: 10 08 2021].

[31] „BumpTech - Glide,” [Online]. Available: https://bumptech.github.io/glide/. [Data uzyskania dostępu: 01 08 2021].

[32] „Dokumentacja Android - LiveData,” [Online]. Available: https://developer.android.com/topic/libraries/architecture/livedata. [Data uzyskania dostępu: 12 07 2021].

[33] „Dokumentacja Android - ViewModel,” [Online]. Available: https://developer.android.com/topic/libraries/architecture/viewmodel. [Data uzyskania dostępu: 05 08 2021].

[34] „Dokumentacja Android - Fragments,” [Online]. Available: https://developer.android.com/jetpack/androidx/releases/fragment. [Data uzyskania dostępu: 12 07 2021].

[35] „AndroidX Tech - Legacy - kompatybilność wsteczna,” [Online]. Available: https://androidx.tech/artifacts/legacy/legacy-support-v4/. [Data uzyskania dostępu: 06 08 2021].

[36] „Makeramen - Rounded Image View,” [Online]. Available: https://github.com/vinc3m1/RoundedImageView. [Data uzyskania dostępu: 30 07 2021].

[37] „Xabaras - RecyclerView Swipe Decorator,” [Online]. Available: https://github.com/xabaras/RecyclerViewSwipeDecorator. [Data uzyskania dostępu: 01 08 2021].

[38] „Stack Overflow - On Swipe Touch Listener,” [Online]. Available: https://stackoverflow.com/a/12938787. [Data uzyskania dostępu: 30 08 2021].

[39] „Stack Overflow - Permissions,” [Online]. Available: https://stackoverflow.com/a/35486162. [Data uzyskania dostępu: 08 09 2021].

[40] „Tatocaster - RealPathUtil,” [Online]. Available: https://gist.github.com/tatocaster/32aad15f6e0c50311626. [Data uzyskania dostępu: 14 07 2021].

SPIS RYSUNKÓW

SPIS TABEL

SPIS ZAŁĄCZNIKÓW

Niniejsza praca posiada załącznik w postaci płyty CD zawierającej wersję elektroniczną niniejszej pracy w formatach PDF oraz DOC, a także kod programu.

STRESZCZENIA

Streszczenie w języku polskim

Praca inżynierska dotyczyła projektu i implementacji aplikacji mobilnej służącej do nauki historii sztuki. Z powodu narastającego problemu przeciążenia informacjami i bodźcami, aplikacja została zrealizowana w sposób unikający wysyłania nadmiaru bodźców oraz informacji swoim Użytkownikom (co oznacza małą ilość komunikatów, absolutny brak reklam, ograniczenie funkcjonalności do potrzebnych i naprawdę przydatnych, łagodną szatę kolorystyczną interfejsów).

Aplikacja została zrealizowana w języku Java, przy pomocy środowiska Android Studio. Wykorzystuje architekturę MVVM wzbogaconą o warstwę repozytoriów. Aplikacja pozwala na zdefiniowanie własnej bazy Obrazków, których dane dotyczą nazwy, autora, lokalizacji, datowania, typu, nurtu i epoki. Typ, Nurt i Epoka należą do elementów periodyzacyjnych, którymi Użytkownik może dowolnie zarządzać. Oprócz możliwości dodawania plików z pamięci urządzenia, Użytkownik może przy pomocy aplikacji pobrać gotowe paczki z Obrazkami. Następnie Użytkownik może uczyć się dodanych Obrazków, a na koniec podjąć się z nich testu, dowolną ilość razy. Dzięki temu aplikacja ma wspomagać swoich Użytkowników w nauce do zaliczeń przedmiotu historia sztuki w liceach oraz na studiach.

Summary in english

In this engineering thesis, a mobile application for History of Art's study was designed and implemented. The problem of the information overload that is increasingly occurring in the modern society was addressed by making the application show solely the information that the User needs for learning the history of art by using flashcards. That refers to: reduced amount of announcements to only the information that User needs for completing his/her tasks, absolute lack of advertisements of any kind, mild colour palette and the amount of functionalities reduced to those that are needed for learning purpose and are helpful.

The app named HaSZ was created using Java programming language and Android Studio IDE. It is based on MVVM architecture, enriched by a repository layer. The app lets its Users to define their own Pictures with descriptions (name, author, location, dating, Type of artwork, art Movement and Art Period) and save them to the database. The Type, the Movement and the Art Period are called periodization items and are under User's full control in the app. Not only can User upload pictures from the device's memory and turn them into Art Flashcards, the app also lets the User download some ready-made packages containing collections of flashcards. Once the User has some Pictures in the app, he/she can proceed to memorizing them and finally take the test (as many times as he/she wants). This is the way the HaSZ application is meant to help students learn for their school / university subject of History of Art's exams.